Noninvasive Methods for Host Certification

PATRICK TRAYNOR, MICHAEL CHIEN, SCOTT WEAVER, BONIFACE HICKS, and PATRICK McDANIEL
Systems and Internet Infrastructure Security (SIIS) Lab
The Pennsylvania State University

Determining whether a user or system is exercising appropriate security practices is difficult in any context. Such difficulties are particularly pronounced when uncontrolled or unknown platforms join public networks. Commonly practiced techniques used to vet these hosts, such as system scans, have the potential to infringe on the privacy of users. In this article, we show that it is possible for clients to prove both the presence and proper functioning of security infrastructure without allowing unrestricted access to their system. We demonstrate this approach, specifically applied to antivirus security, by requiring clients seeking admission to a network to positively identify the presence or absence of malcode in a series of puzzles. The implementation of this mechanism and its application to real networks are also explored. In so doing, we demonstrate that it is not necessary for an administrator to be invasive to determine whether a client implements required security practices.

Categories and Subject Descriptors: K.6.5 [**Management of Computing and Information System**]: Security and Protection

General Terms: Security

Additional Key Words and Phrases: Certification, assurance, network security, malware

ACM Reference Format:
Traynor, P., Chien, M., Weaver, S., Hicks, B., and McDaniel, P. 2008. Noninvasive methods for host certification. ACM Trans. Inf. Syst. Secur. 11, 3, Article 16 (March 2008), 23 pages. DOI = 10.1145/1341731.1341737. http://doi.acm.org/10.1145/1341731.1341737.

Based on “Noninvasive Methods for Host Certification” by Patrick Traynor, Michael Chien, Scott Weaver, Boniface Hicks, Patrick McDaniel which appeared in *Proceedings of 2nd International Conference on Security and Privacy in Communication Networks (SecureComm)*, Baltimore, MD © 2006 IEEE.
We wish to acknowledge the support of CISCO through the University Research Program (URP). Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of this sponsor. We would also like to thank Divya Muthukumaran, Joshua Schiffman, and our anonymous reviewers for offering insightful comments.
Author’s address: P. Traynor, SIIS Lab, Department of Computer Science and Engineering, The Pennsylvania State University; email: traynor@cse.psu.edu.

1. INTRODUCTION

Access to and protection of many networks have traditionally been predicated on user authentication. Users providing the necessary credentials, typically in the form of a password, are given access to some authorized subset of resources within a domain. While separating "insiders" from unauthorized users, common network admission processes make no assessment of the safety of the connecting host. Because the machines of even the most trusted users are becoming the unwitting hosts of the current malware pandemic, user authentication alone is no longer a sufficient mechanism for providing protection. Instead, the security configuration of machines must be inspected to ensure that all hosts are appropriately protected.

Vetting hosts in this way is a problem of *certification:* Untrusted hosts need to be evaluated to ensure they meet some predefined best practices for security. Once a host is determined to meet these minimal standards, it may be allowed access to the network or be subjected to further inspection or authentication. The set of practices one must adhere to is a critical and environment-specific element of a network's security policy.[1] However, the manner in which proof of adherence to these practices is obtained is a thorny issue. Because simply asserting an inherently untrusted platform's settings is insufficient, more invasive mechanisms are typically employed.

Given the available techniques, system scanning is the most often used form of host vetting. Before allowing machines to log on to a network, organizations such as the NSF require that they first be aggressively scanned for malware and security configuration by specialized software. Commercial software packages implementing system scans are widely available [Eustice et al. 2003; Sygate Web site; Symantec; Zone Labs]. While effective, this approach is neither desirable nor scalable. While the NSF may be trusted to scan laptops, requiring such exposure of personal or proprietary data may not be acceptable in more general settings.

Evolving access patterns only exacerbate the challenge of privacy-retaining certification. For example, wireless hot spots may require all hosts to be scanned for viruses prior to being permitted access. These and other procedures may legally indemnify a service provider from damages resulting from viruses or other malcode [Harris 2004; Olsen 2002]. Such requirements are totally appropriate and are likely to be observed more frequently in the future. However, allowing the proprietor of a local Web cafe to run arbitrary scans and code on the customers' laptops, many of which likely contain sensitive financial information, is unacceptable. We argue that such trade-offs need not be absolute. Specifically, the average user need not relinquish his or her privacy in the process of demonstrating correct adherence to security policy.

In this article, we develop and evaluate a noninvasive host security certification procedure. As a means of demonstration, we introduce a protocol

[1]A formulation and definition of network access best practices is explicitly outside the scope of this work. Readers interested in guidelines in this area should consult CERT or other professional security organizations.

that allows a host to prove that it has properly configured up-to-date antivirus software without any direct access to its internal state. Inspired by so-called cut-and-choose protocols, including zero-knowledge proofs [Goldwasser et al. 1985], the network provides each client with a vector of randomly selected file blocks, which are either malcode or harmless placebos. The client is certified if it can identify which blocks contain malcode. Hence, the host proves the existence of correctly operating malcode detectors by demonstration. Any host that cannot differentiate malcode from placebos in the test vector is deemed unsafe and disallowed access. As is appropriate for the target environments, our model assumes that the user is not intentionally malicious but may be an unwitting carrier of malware or may have outdated antivirus software. We implemented our protocol using COTS malware detectors and tested it in a live network environment. The efficiency and provided assurance are evaluated, and we comment on the challenges and operation of our noninvasive certification tools.

Note that our approach can also be generalized to handle many forms of security infrastructure—any security mechanism that is able to distinguish between normal and malicious behavior may be nonintrusively verified. The contribution of our work lies not only in the design of the protocol but also in the efficiency, flexibility, simplicity, and nonintrusive nature of its implementation.

The remainder of this article is organized as follows: Section 2 offers a brief overview of the relevant related work; Section 3 defines adversary and protocol models; Section 4 discusses the implementation of this system and examines the performance of this protocol; Section 5 discusses the results and the application of this protocol to real systems; and Section 6 provides concluding remarks and future avenues for this work.

2. RELATED WORK

The authentication of users is often a necessary prerequisite for access in many public networks. For example, Needham and Schroeder [1978] presented a widely used protocol to authenticate and initiate communications between two machines. Kerberos improved upon this work by extending the protocol and building an authentication service upon it [Steiner et al. 1998]. More recent research in authentication includes the use of smart cards [Kato et al. 2003; Kawase et al. 1998; Scheuermann 2002] and biometrics [Rahman and Bhattacharya 2003]. Generally speaking, user authentication prevents unknown or unwanted entities from accessing a system. While this is necessary for some environments, it is not sufficient to determine the security of the joining host; authentication validates identity but states nothing about the configuration or state of the authenticated party's host.

To address this problem, some networks require that each new client be scanned by an agent in the network. This requires running foreign code on the machine—a practice that both violates privacy and is also insecure. Eustice and colleagues [2003] use available software packages to verify the state of a mobile computer user in the QED protocol suite. Users are first isolated from the network and subjected to a system scan. Systems failing the scan are required to update with patches or prompted to turn on required software. While this approach does not require the use of specialized hardware or software on the user's computer, it does raise privacy concerns.[2]

Attestations of software are actively being explored as a solution for noninvasive host certification [Buldas et al. 2002; Garfinkel et al. 2003; Sailer et al. 2004]. In one instance, the Trusted Computing Group's trusted platform module (TPM) vouches for the current state of a system by cryptographically certifying the precise operating system and application software running on a system. There is hope that, through future features, the TPM will offer unforgeable verification of a platform's ongoing safety. However, such features will not solve the security requirements of legacy devices. Moreover, even if features were available, they could only attest to the software that is running—nothing prevents software from being loaded after an attestation occurs. Hence, the guarantees provided by such devices are not necessarily complete.

Inspired by the vast body of literature in zero-knowledge proofs [Aronsson 1995; Goldwasser et al. 1985], we take a different approach than the historical authentication, scanning, or attestation methods. Zero-knowledge proofs allow a prover to demonstrate knowledge of an artifact (typically secret information such as a key) without actually exposing it. Such proofs are typically implemented using considerable cryptographic machinery and are used as the basis for many valuable constructions. Using an analogous noncryptographic approach, we ensure that the host to be certified (prover) demonstrates, with a high degree of accuracy, that it is running recent and well-configured antivirus software. This mirrors a zero-knowledge proof in that the host proves it is executing the software without allowing the certifying party any direct access to the software's execution environment.

Our approach to remotely ensuring one aspect of the secure configuration of a machine is complementary to past certification methods. Note that all of these above methods have significant tradeoffs and limitations. Attestations allow certification of the running software. Scanning by a third party affords deeper inspection and hence stronger validation of the current state of infection but says nothing about the platform's ability to protect itself against future incursions. Each of these methods is appropriate for a set of environments. None of them is sufficient for the environment that is becoming rapidly most common: public access points to the Web. This is an environment of mutual distrust, in which service providers cannot fully trust their clients and just let anyone log in, nor can clients merely allow service providers full access to all their files. This environment presents the particular challenge that it should be accessible to all users and yet it should also not compromise the privacy of those users. Our central contribution is providing an additional tool for host certification that is appropriate for these environments of mutual

[2]Eustice and colleagues [2003] declare, "In some environments, safety must take precedence over privacy."

distrust: This tool can certify the security of foreign host clients while also not violating their privacy.

3. PROTOCOL DEFINITION

In this section, we present our procedure for certifying the proper functioning of antivirus software on a client machine. In the subsections that follow, we describe the adversary, state our design goals and define the protocol itself. We complete this section with a discussion about the selection of specific malcode.

3.1 Defining the Adversary

The success of the most damaging worms and viruses sweeping across the Internet has depended largely on their ability to quickly infect a multitude of unprotected machines [Staniford et al. 2002]. Indeed, the number of platforms unknowingly spreading a digital contagion is far greater than the number of people who are actively and consciously attempting to further its diffusion. Accordingly, the most prevalent (and arguably the most dangerous) threat to any network is the presence of a susceptible or infected platform. *Therefore, the "adversary" for whom this work is designed is the authorized but unaware user.* More specifically, we assume that the user of the host is not intentionally trying to infect others. We require that the user be able to prove, with a high degree of certainty, at the point of certification, that he or she has a properly functioning security infrastructure. That is, we desire to prevent an adversary from claiming that it has an up-to-date, well-configured infrastructure when, in fact, it does not.

The threat model adopted in this work mirrors the standard model in this area. Much of the work conducted in this field has suggested the need for active scans of machines as they attempt to attach to a network [Eustice et al. 2003; Sailer et al. 2004; Symantec; Zone Labs]. The goal of these scans is to detect and clean malcode from infected platforms. This is typically accomplished by examining the contents of all the files on a machine and comparing them against known malcode. Such measures serve only to protect the network from a machine at the time of certification—there are countless ways in which a truly malicious adversary could circumvent this defense. For example, the malcode intended for release can simply be encrypted or stored on removable media and therefore remain hidden from detection. Worse yet, there is nothing to prevent an adversary from successfully passing a full system scan and then downloading the malignant payload from a remote server. Full system scans by network administrators are simply unable to prevent attacks by a determined adversary.

There are numerous settings in which such a threat model is appropriate. As discussed previously, the administrator of a wireless hot spot may require clients to demonstrate that they are able to protect themselves. Such an approach is also reasonable for when an individual has the expressed obligation to prevent such an administrator from potentially accessing his or her system. For instance, external account auditors, legal counsel, and employees of companies dealing with sensitive information (e.g., credit card vendors, banks, brokerage firms) should be able to connect without needing to disclose the full content of their hard drive.

To repeat, this protocol is designed to protect networks against well-intentioned users who *unknowingly* have machines that either are vulnerable to or already infected with malcode.

3.2 Design Goals

The design goals of this protocol are *preserving user privacy, flexibility, simplicity of use,* and *efficiency*. In terms of privacy, this protocol allows for a client host to demonstrate that it exercises good security practices without requiring invasive searches and scans. Secondly, to have an impact on the widest possible clientele, such a solution must be flexible enough to remain agnostic to any specific antivirus software suite. Thirdly, requiring active user participation in any protocol must be kept to a minimum for any solution to be accessible to the general public. This not only precludes requiring the user to respond to messages but also must avoid mandating that a user install one specific software suite to be compliant. Finally, this protocol must perform with reasonable efficiency.

The first two of these goals are achieved by the design of the protocol itself, which takes advantage of features available by default in the majority of available antivirus programs. To support the remaining two goals, our software has been implemented as a Java applet. This applet appears on the terminal when the client attempts to log into a network via a Web-based access system, and it hides all of the underlying functionality and messaging associated with the zero-knowledge inspired client puzzle protocol. This applet displays the client's current status and alerts him or her of successful and failed attempts to attach to a network. We detail the cost and operation of this process below.

While we stress its applicability to Web-based gateways, this mechanism is not limited to only these systems. For example, in a network where authentication is not necessary, a DHCP server could run this process prior to assigning IP addresses. In a more controlled network, protocols such as 802.1X [Congdon 2003] could also be modified to include such a client puzzle access control mechanism prior to user authentication. Moreover, certification need not be used to regulate network access only; many providers of sensitive content may require some kind of certification before distributing data. Independent of the purpose or environment, the operation of the protocol will be quite similar.

3.3 Protocol Definition

We begin the discussion of this protocol by defining the notation used throughout:

—C, S are principals, specifically client and server.

—F_X is a file transmitted by S to C.

—$H(F_X)$ is the hash of the file F_X.

—I_N is the calculated infection index.

—N is a nonce.

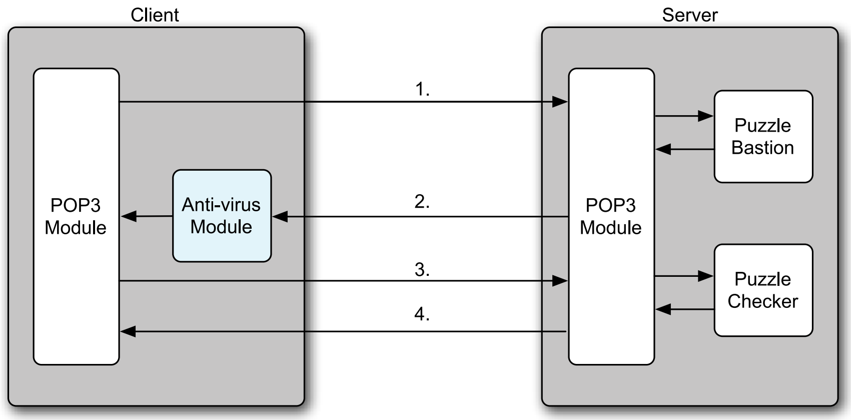

Fig. 1. The protocol for demonstrating a properly functioning antivirus service begins with a client (1) sending a request to the server to join the network. The server (2) responds with a test vector, created by the test generator, which contains a random mixture of clean and infected files. It then awaits a response. After the client's antivirus software intercepts and scans the test vector, the client (3) reports back to the server with a list of which files were clean and which were infected. The server (4) then admits or expels the client based on whether or not the client correctly identified the malcode in the test vector.

—P is the number of files used per round.

—R is the number of rounds C must correctly pass to be admitted.

—$REQ_{X(C,S,N)}$ is a request from C to S, where X is either J for "join" or A for "answer."

—$REP_{X(C,S,N)}$ is a join request response message from S to C, where X is either A for "accept" or F for "failure."

The certification protocol works as follows: A client attempting to attach to a network sends a join request to the server. The server responds by sending the client a test vector—a collection of files that must be scanned and categorized as infected or clean by the client. Note that to remain agnostic to any particular antivirus suite, we do not ask the client to report the name of the malware; rather, we use the repetition of detection as a means of assurance. The client responds to the server with a string representing the infected/clean pattern of the original test vector, which is compared against the precalculated answer at the server. If the two strings match, the client is admitted. If there is any deviation in the strings, the client is refused entry to the network. Figure 1 illustrates this protocol, which is described in further detail below.

A client wishing to attach to a network initiates communication by sending a join request, $REQ_{J(C,S,N)}$, to the server via the POP3[3] protocol. Because the protocol is run over POP3, the antivirus module automatically scans all incoming packets. This is a feature available in all of the major free and commer-

[3]Note that our use of POP3 is tailored to current antivirus tools. Other protocols, for example, HTTP or SMTP, could be readily used for our purposes to the same effect, except that other antivirus mechanisms would be used to vet the incoming data and identify malware.

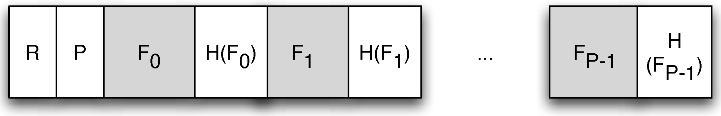

Fig. 2. A test vector message from the server to a request for network access. Each packet contains the number of successful rounds R needed for entry, the number of files P per test vector, and test files F_0 through F_{P-1} coupled with their corresponding hash values.

cially released antivirus software suites. We take advantage of the fact that this mechanism is turned on by default in all of these products. We leverage this default setting to perform the necessary scanning of test vectors. (Methods of ensuring that the user maintains the proper security configuration during the entirety of the attachment period are discussed in Section 5.3.)

As shown in Figure 2, the server responds to the request with a message containing the number of rounds R that must be passed for admission, the number of files P to scan per round, and a test vector. The puzzle itself is assembled by the puzzle creator. Each vector consists of P files, F_0 through F_{P-1}, and their corresponding hash values, $H(F_0)$ through $H(F_{P-1})$. A bit string, I_N, representing the infection status of the files, is then created and kept for the confirmation of the client's response. For example, a sample puzzle containing five files may be represented by 10110, where 1 and 0 represent infected and clean files, respectively. The test generator then batches the components mentioned above, minus I_N, into a MIME message and forwards the data to the client. This protocol is summarized in Figure 3; its performance is detailed in Section 4.

When the client receives the response from the server, the antivirus module is automatically triggered by the use of the POP3 protocol. Only after the test vector has been completely scanned by the antivirus module (and declared safe) is it delivered to the client's POP3 module. Note that because the antivirus steps in between the client and server while infected files are in use, there is no risk of contamination of the client's machine. Additionally, because the files themselves are never executed, the client's platform is at no risk of being infected. To prevent damage, in the case that antivirus software is turned off, the buffers containing the files themselves are zeroed after each iteration. Because files that contained malware have been altered (cleaned or deleted) from their original state by the antivirus module and therefore no longer match their original hash, determining the infection status of each of the files in a vector is as simple as comparing two hash values of the files before and after the antivirus detection procedure is executed.

As is done in the test generator, the client creates the corresponding infection index I'_N for the received files. The infection index is returned to the server, which passes the client's solution to the answer checking module. The module performs a lookup for the infection index calculated by the puzzle creator and compares the two values. A success or failure message is returned to the server's POP3 module. The module then either sends the next puzzle, if necessary, or informs the client of its correct response and allows it to enter

$$\begin{aligned}
&1)\ C \rightarrow S : REQ_{J(C,S,N)} \\
&2)\ S \rightarrow C : R, P, N, F_0, H(F_0), F_1, H(F_1), \cdots, \\
&\qquad\qquad F_{P-1}, H(F_{P-1}) \\
&3)\ C \rightarrow S : REQ_{A(C,S,N)}, I'_N \\
&4)\ REQ_{X(C,S,N)}
\end{aligned}$$

Fig. 3. The challenge/response protocol used to determine the presence and proper functioning of antivirus software on a client's platform.

the network ($REP_{A(C,S,N)}$) or reports the client's failure and ends the session ($REP_{F(C,S,N)}$).

3.4 Selecting Malcode

Without the proper selection of malcode for a specific environment, the value of this protocol is limited. A carefully chosen sampling of viruses, however, can be used to demonstrate that a system is protected against the most critical digital pathogens with a given assurance. The challenge in selection comes in ensuring that the files sent to a client are not only representative of the malcode most likely to affect a specific network but also that the chosen malcode appears sufficiently random so as to make guessing the infected files extremely difficult.

We begin by first defining the term assurance. As the number of files in a test vector increases, a client correctly classifying an entire test vector is probabilistically more likely to be running antivirus software. The probability that a client is able to guess the status of infection for an entire test vector therefore decreases exponentially with the size of the test vector. For example, while a client presented with a single file can correctly guess whether or not the test file is infected with a probability of 0.5, a client receiving ten files is able to guess the correct Infection Index with a probability of only $9.76 * 10^{-4}$.

Throughout the remainder of this text, we refer to the level of assurance according to the standard engineering metric of "nines." The example client above that correctly categorizes all ten files in the test vector is said to be running antivirus software with three-9s (0.999) probability. Assurance is simply $1 - Pr[\textit{Guessing Correctly}]$.

Using Equation 1, it is possible to determine the number of files P necessary to achieve a desired assurance. Assurance levels of three-, six-, and nine-9s are accordingly achievable through the use of 10, 20, and 30 files, respectively.

$$P = \left\lceil \frac{\log(1 - \textit{Assurance Level})}{\log\left(.5^{R}\right)} \right\rceil \tag{1}$$

We now explore various methodologies for selecting specific instances of malcode for use in test vectors. An important observation in selecting malcode is that the majority of machines connected to the Internet are eventually patched or upgraded. That is to say, malcode becomes less effective at infecting machines over time. The malcode that wreaked havoc a decade ago is highly

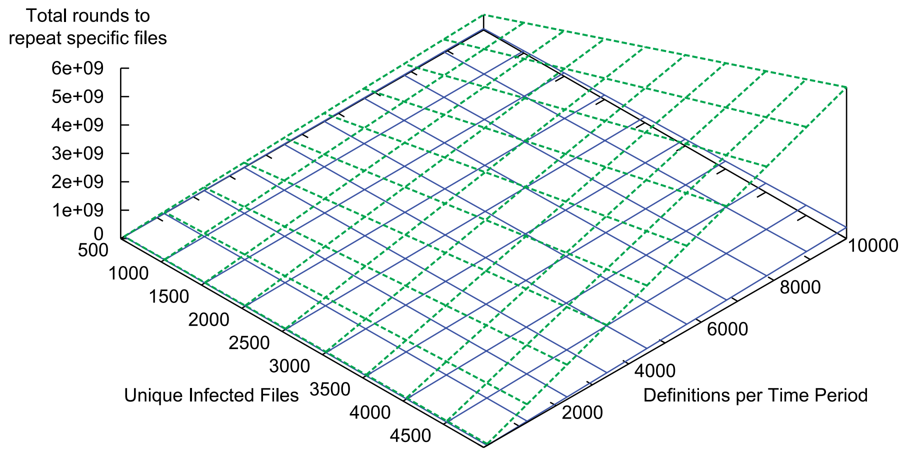

Fig. 4. Examination of the trade-offs between the number of viruses per time period (based on a geometric distribution), the number of unique files in which each virus is manifest, and the total number of rounds necessary to encounter a given file twice. In such a system, the frequency of replay is inversely proportional to the space allocated for storing test files.

unlikely to be able to establish a foothold in today's machines. While it is impossible to assume that these malicious programs ever completely vanish, the probability of being infected with a given piece of malcode decreases as a function of time. Additionally, because of the rapid spread of modern malcode [Staniford et al. 2002], the likelihood that a machine is infected with one of any number of recently released digital pathogens is much more probable.

Time is therefore a natural metric for pathogen selection; newer viruses are more likely to be encountered in the wild, and hence it is desirable to test protections against them more frequently. From the discussion above, however, simply selecting the most recently released malcode or randomly selecting one from a uniform distribution is not sufficient. A more robust system achieves both recency and breadth by choosing malcode based on a number of realistic models of decay. The most frequently used models for both digital and biological pathogen lifetimes use exponential [Bailey et al. 2005] and geometric [Garetto et al. 2003] decays.

In addition to the choice of distribution, the effects can also be tuned by adjusting the size of time periods. Most commercial antivirus programs, for example, release weekly updates of malcode definitions. Accordingly, the smallest time period for time-based models in our system is one week. This level of granularity is helpful for ensuring that client platforms are updated to the most recent set of definitions. Because the number of viruses per update is limited, extending the length of time periods may instead prove more valuable as the frequency of replay (and therefore the ease of guessing) is decreased. The trade-off between recency and replay must therefore be carefully balanced. Figure 4 illustrates that the trade-offs between recency, breadth, resistance to guessing, and storage can be set depending on the specific distribution used.

Time is a natural metric, but it is not the only means of calibration. For example, it may also make sense for an administrator to base defenses around virulence. In this case, test vectors should include tests for more dangerous viruses with the greatest frequency. If the majority of malcode in recent virus signature updates do little damage to actual systems, it is difficult to justify that these pieces of malcode should be tested for more frequently. Accordingly, our tool could be tuned to take advantage of virulence classifications provided by many antivirus software providers. Using this information, the proper statistical distributions could be formed and a network could be fine-tuned to prevent the most catastrophic malcode with higher assurance. Continuing to maintain a broad sampling of test files helps to preserve breadth as well.

One final approach that could be used is based on the protection of specific services. Networks providing particular services to customers may choose to protect against malcode specifically designed to interrupt that service. For example, a network composed primarily of Microsoft SQL servers may be primarily concerned with blocking exploits designed to attack this application. Like the previous techniques, a system tuned in this manner should still provide breadth against other attack vectors, as the primary application running on these systems may not be the only means of exploiting vulnerabilities.

Of course, all of these approaches can also be used in concert to provide additional protections. A system based on time could subdivide each quantum into distributions based on virulence. In turn, a system built to protect primarily against a particular family of attacks could select these exploits most frequently but fill in the remaining test vector slots based on a temporal method. These additional protections come at the additional cost of classification, setup, and maintenance.

The tuning of resilience must be carefully considered and set by the administrator of any system to accurately protect against the most relevant adversary.

3.5 Selecting Placebos

Addressing the selection of malcode represents only half the problem. To illustrate this point, we examine the use of text documents as a source of uninfected files. On receiving the test vector, an adversary could scan each incoming file for dictionary words. Those files composed largely of human-recognizable content could be discounted as benign simply by inspection. The files remaining after such filtering would, with high probability, represent the malicious code sent by the server. This simple example demonstrates an important point—for the results of such testing to be of any value, the difference between infected and "placebo" files must be imperceivable. If the uninfected elements of a test vector are easily discernible from their infected counterparts, the strength of the guarantees offered by this system would be greatly diminished. While an adversary actively attempting to subvert the system is explicitly outside our threat model (see Section 3.1), we attempt to make the protocol as robust as possible through such considerations. Specifically, a client with no malicious intent may simply not wish to pay for antivirus software or endure the overhead of such operations.

As an alternative to text files, a combination of user and system binaries may achieve the desired level of protection. For example, a selection of files from sources as varied as Web content caches, temporary files created during the compilation of user code, or samples from system libraries (e.g., .DLL or .so) would make inspection attacks much more difficult. Unfortunately, the use of such files may create a more significant problem—information leakage. Unencrypted financial or personal information may inadvertently be offered to any party attempting to log into the system. Moreover, the browsing habits of the users may themselves be sensitive. File header information, well-known formats, or partial images may allow parties engaging in corporate espionage to gain insight into a target's strategic plans or simply offer the client a clue to the maliciousness of the payload. Temporary files created during the compilation process may cause similar problems. Likewise, system binaries or libraries may inadvertently provide an intentionally malicious adversary with information useful in compromising a target (e.g., the presence of unpatched software). While it is possible to use files from other systems, maintaining a large enough collection of unusable files may become expensive. Such approaches would therefore require significant filtering to avoid the inadvertent exposure of sensitive information.

A third option is to use randomness in the creation of placebo data. Pseudorandomly generated content avoids many of the previously mentioned pitfalls and eliminates the need for egress content filtering. However, the use of specific pseudorandom functions carries a variety of assumptions and trade-offs. Generators including the C language's `stdlib rand()`, which introduce minimal computational overhead, are in fact deterministic and often easily guessable [Bellovin 1989]. Functions such as OpenSSL or `/dev/random` can offer improved sources of pseudorandomness at the price of decreased performance. As is discussed in Section 4.2, the use of external computational resources (e.g., a puzzle "bastion" [Waters et al. 2004]) can alleviate most of this impact on performance. Unfortunately, the exclusive use of placebo files created from even a "good" source of pseudorandomness may in fact weaken the protection of the overall system. Through the use of simple static analysis techniques [Chinchani and van den Berg 2005], the absence of program control-flow structures (i.e., sequences of legitimate assembly instructions) frequently found in malcode would potentially allow an adversary to eliminate many of the elements in a test vector.

Recognizing this, a final technique for generating uninfected files would be through the creation of "plausible" sequences of instructions. As often observed by the intrusion detection community [Toth and Kruegel 2002; Wang and Stolfo 2004], certain classes of attacks are frequently recognizable by the presence of assembly code. Accordingly, producing a short series of assembly commands (e.g., `add`, `push`, and so on) accessing similar portions of memory could be used to create credible attack vectors to an adversary performing simple analysis techniques. This solution would be the most robust against this more broadly defined adversary (i.e., not malicious but unwilling to run AV software), as execution of such sample code would not provide a conclusive indication of its maliciousness. To avoid misclassification and potentially to reduce false

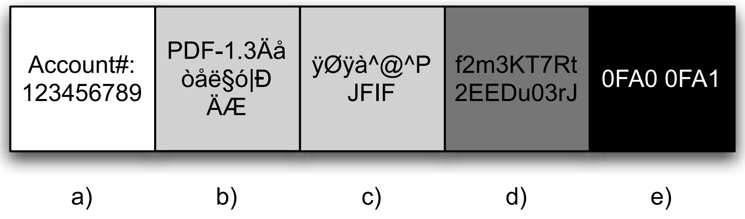

Fig. 5. Examples of uninfected files for use in a test vector. Sample (a) represents cached Web data and may overtly leak sensitive information. Examples (b) and (c), which represent PDF and JPG files, may expose user browsing habits or confidential internal data. File (d) is the base-64 encoded result of OpenSSL's random function [OpenSSL]. Example (e) can be interpreted as stack push and pop commands for the FS segment register in the x86 IA-32 architecture [Intel Corporation 2006].

positives, such files could be tested against known attack signatures [Paxson 1999; Snort]. Placebos accidentally marked as malware prior to transmission can then simply be discarded. Like the previous technique, the creation of such strings may be made more efficient through the use of additional computing infrastructure.

Given the above considerations, the most effective approach for generating uninfected files is likely a mix of the above techniques. A combination of content from white-listed Web sites (e.g., news, sports and the like), random data, and plausible sets of assembly code would create the most robust defense against inspection attacks. In the absence of such a mix, however, the final method is likely to provide the highest amount of indistinguishability for placebos. Figure 5 offers an overview of these techniques and their trade-offs.

4. PERFORMANCE EVALUATION

In this section, we evaluate the efficiency of the certification process. We begin with some notes on the construction of our tool. The Java programming language was chosen to implement the protocol, allowing us to execute the entire process in a sandbox. Because the overwhelming majority of viruses are written for Windows-based systems, we believe that demonstrating the performance of this procedure on a Windows box is essential. Accordingly, the server runs on a 1.6 GHz Pentium 4 Windows XP Pro version 2002 SP1 with 256 MB RAM. The client runs on a 1.6 GHz Pentium 4 Windows XP Pro version 2002 SP1 with 256 MB RAM running Symantec Norton AntiVirus. We select this particular antivirus software because it represents almost two-thirds of the market share [Evers 2006]. We used a Netgear 10/100 Dual Speed Hub between the server and the client on a 100MB/sec network. To prevent the server's resident antivirus software from overzealously cleansing the infected virus test files, the server selects from the 3,335 malware files located on a CD-ROM. The malcodes themselves averaged between 2KB and 4KB in size. A test vector containing 30 "puzzles" can therefore easily be delivered using approximately 100KB—a relatively insignificant amount of bandwidth for the

Table I. Microbenchmark Results for Zero-Knowledge Client Puzzles

Operation	$\bar{x}$ Time (ms)	Std Dev (σ)
Network	0.1	1.9
Antivirus Scanning	425.2	48.2
MIME Parsing	1.0	3.5
Hashing	1.3	3.8
Total Time	431.5	58.9

majority of access points.[4] Uninfected files of sizes similar to the above malcode were generated using `rand()`.

For completeness, additional tests of the client program were conducted on a 930 MHz Pentium 3 SUSE Linux Box with 256MB RAM running RAV AntiVirus. Because the results on both systems were similar, the results discussed below reflect only the experiments conducted on the Windows machine. The implementation of the protocols on the client and server occupied approximately 80KB of disk space. Notably, the programs required no additional software to be installed prior to joining a network, and no caching between certifications was necessary. In accordance with our design goal of efficiency, the software had no persistent on-disk or memory footprint for the client and so required no sustained resources on the host. For increased safety, we could relax these requirements slightly and introduce tickets, as discussed in Section 5.3.

4.1 Microbenchmarks

The results of our microbenchmark tests of this system are shown in Table I. Each of the 10,000 iterations included a single round consisting of a test vector containing 30 files. This test vector corresponds to nine-9s assurance that a client is running antivirus software. The microbenchmarks measure the time spent on network transmission (*Network*), scanning (*Antivirus Scanning*), MIME encoding and decoding (*MIME Parsing*), and SHA1 hashing of the received and potentially cleaned files (*Hashing*).

As expected, the dominant factor in the execution of this protocol was the scanning of incoming files. This activity, responsible for over 98.5% of the execution time over the average 431.548 milliseconds needed to execute this protocol, takes two orders of magnitude more time than all of the other operations combined. The observed variance can be attributed to the different file sizes in each puzzle and transient network and operating system events.

4.2 Macrobenchmarks

While the microbenchmarks tested a 30-file test vector in a *single* round, there are a number of compelling reasons for executing such a protocol across *multiple* rounds. For example, because it is significantly faster to transmit files than to scan them, it may be more efficient to issue puzzles with fewer individual files and test them in smaller batches over a larger number of rounds. In this way, we could maximize performance by exploiting the relative speeds

[4]Wireless networks running 802.11b or g protocols can transmit at rates of multiple Mbps—beyond sufficient bandwidth to support our protocol.

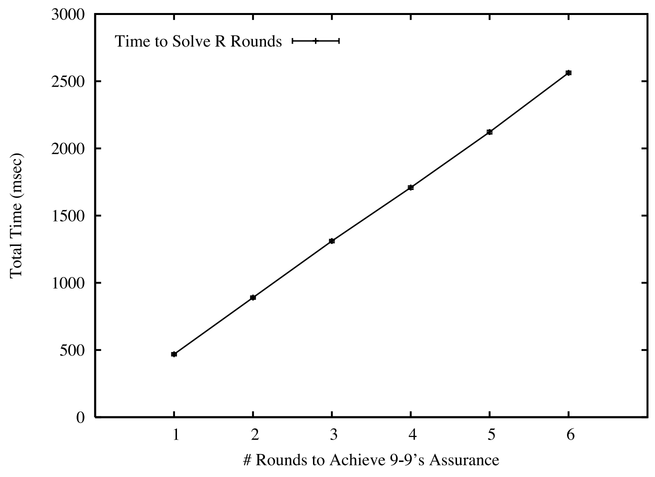

Fig. 6. The amount of time required to yield nine-9s assurance (30 files total) over a varying number of rounds. Note that even though the number of files scanned per round is inversely proportional to the number of rounds, the increase in rounds is highly linear.

of transmission and scanning to achieve a pipelining effect. Moreover, such a mechanism could decrease the total number of files transmitted because a failure would short-circuit the transmission of the remaining files. In testing this hypothesis, however, we discovered that this was not an effective approach.

Figure 6 shows the correspondence between execution time and the number of rounds required for the delivery of 30 files in the puzzle (thereby giving nine-9s assurance). Each data point was collected from 1,000 iterations of the protocol. The observed increase in time with respect to the number of rounds is almost perfectly linear. The average change in cost in Figure 6 averages 418.77 milliseconds. As demonstrated above, the dominating factor responsible for this behavior is the scanning of incoming test vectors. Further investigation revealed that the cost of scanning files is dominated by the start-up cost of the scanner. That is, the scanner is not memory resident, and a new process is started each time a new batch of files is received.

Figure 7 demonstrates further evidence of the start-up cost dominance. The number of files, and therefore the level of assurance, is varied between three- and nine-9s. As the level of assurance increases, so too does the mean cost by an average of 4.0 and 4.4 milliseconds for one and two rounds, respectively. Through linear regression, the cost of adding an additional order of magnitude of assurance x can be described by the functions $y = 4.03x + 431.1$ and $y = 4.85x + 848.4$. The running time in milliseconds for R rounds at any assurance level (number of "nines") can therefore be approximated as:

$$Time(ms) = 418.77R + (4.44 \times Assurance\ Level). \tag{2}$$

The actual scanning of files therefore has little effect on the overall time required for the protocol to operate; rather, the running time is predominantly determined by the overhead associated with starting the scanner at the beginning of each round.

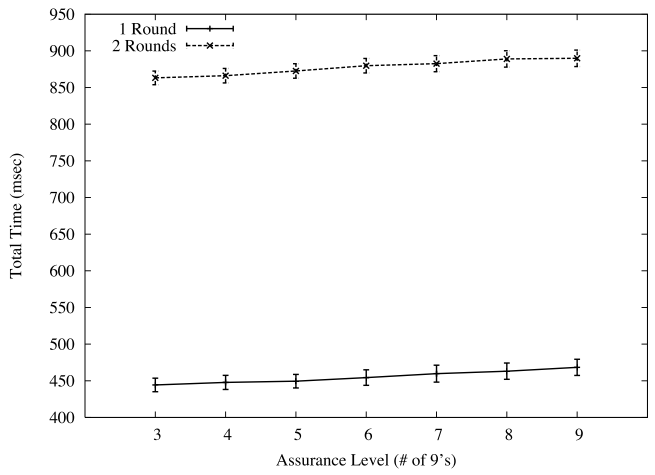

Fig. 7. Macrobenchmark performance results for varying levels of assurance. As is observable in Figure 6, the difference between a single versus multiple rounds of puzzle solving is dominated by the antivirus scanning portion.

Although it is clearly less efficient, there may still be reasons that it desirable to use multiple rounds. For example, it may be undesirable to force the server to deplete resources to access more files than necessary. If a system can be proven insecure, it saves server resources to discover this with smaller test vectors. Fortunately, there are methods that can be used to decrease the reliance on smaller test vectors while still preventing resource exhaustion (i.e., puzzle depletion) at the server. Waters and colleagues [2004], for example, study a similar problem for cryptographically based client puzzles and are able to leverage the use of a "bastion" to generate a high number of puzzles on behalf of the server. However, the cost of assembling puzzles is likely to be sufficiently low such that requiring off-line puzzle-generating bastions may not be warranted in all but the most active areas.

While explicitly outside our threat model (see Section 3.1), a malicious adversary may try to circumvent the system by extracting and storing test vectors.[5] Specifically, the ability to determine which guesses are correct and which are incorrect increases dramatically as the size of a given test vector decreases. For a test vector with a single file, for example, a client will *always* learn whether that file was malcode. This realization, combined with the results regarding efficiency, gives compelling evidence that the number of rounds used should be kept low. At the same time, we leave our system flexible enough that this parameter could be adjusted according to the needs of a particular network.

Similarly, an adversary who promiscuously listens to the protocol exchanges of many joining hosts may create a dictionary of puzzles by looking at the server-provided puzzles and client-provided responses. Hence, that adversary

[5]Such behavior may be exhibited by an adversary unwilling to purchase antivirus software.

would be able to produce a correct answer with some probability, depending on the number and diversity of exchanges that were eavesdropped. Again, such adversaries are specifically outside our threat model, but we note that any environment wishing to thwart such attacks need only use some transport level security protocol such as TLS/SSL [Dierks and Allen 1999; OpenSSL]. Such protocols are currently supported by most mail applications, and hence their integration into the process described above should be straightforward.

Our experiments with both Norton AntiVirus and RAV can potentially be viewed as an upper bound on the performance of such a protocol. For instance, if antivirus software were to be constantly kept in memory, as opposed to being forked in response to specific messages, the cost of executing a round of the protocol would be significantly reduced. While such behavior is possible, it is not likely to exist in a significant number of antivirus suites. Primarily, the infrequent arrival of POP3 messages would mean that such a design decision would require a device to devote additional resources to keeping a generally unused mechanism in memory. Because most antivirus software already has a large associated overhead, such a design decision would be unlikely to be implemented. Regardless of whether the scanning routine is stored constantly in memory or loaded as needed, these results demonstrate that our protocol can be executed without an undue burden on the resources of a client machine.

5. DISCUSSION

In the remaining subsections, we compare the performance of our implementation to the work of others in this area, discuss methods of increasing the security guarantees provided by such a protocol, examine the inherent safety issues to this particular implementation, and conclude with a sampling of additional applications to which a similar approach could be applied.

5.1 Certification Performance

Comparing our approach with other certification processes is not straightforward. For example, performing a full scan of a system, as is suggested in Eustice and colleagues [2003] may take time on the order of tens of minutes and is highly dependent on the amount and content of disk space occupied by a client. By contrast, our protocol takes a relatively short, fairly constant amount of time to execute. Of course, the goals and level of assurance provided by each approach differ vastly, so any performance comparison would be of limited value.

Client-puzzles enforcing rate limitation on clients [Juels and Brainard 1999; Waters et al. 2004] can be varied in difficulty and be made to last anywhere between minutes and seconds. Our protocol, too, can increase the difficulty required to gain access to a system; however, the orders of magnitude between rate limiting solutions and process verification are necessarily different. Attestation-based solutions, which may offer the most comparable solution, also cost up to tens of seconds [Brickell et al. 2004; Sailer et al. 2004], even when implemented in hardware (e.g., TPM).

5.2 Infection

The security of the protocol discussed in this paper is entirely based on the accuracy of the antivirus software used by the server. Logically, if the server's antivirus software is unaware of a particular virus (e.g., a zero-day exploit), our protocol will also be unable to defend against this pathogen.

Should such a virus take control of a device, the potential also exists for it to parrot the operations of our protocol. For instance, the virus could keep its own database of exploits and perform the scanning behavior as normal. Being able to correctly answer the puzzles it receives, this malcode could ensure that a device was admitted to the network without actually running antivirus software. Such a strategy, while likely successful in the short term, would not be sustainable in the long term. Once an update became available, such a virus would be removed.

Devices infected with a rootkit present new challenges. Because rootkits can prevent antivirus software from functioning properly or from scanning certain files, additional protections may be necessary. We leave solutions addressing this specific issue to future work.

5.3 Maintaining Compliance

While the protocol presented thus far demonstrates, for a required level of assurance, that a client is running antivirus software when logging on to a network, there is no guarantee that the client will continue to do so after the initial test. A user may disable the antivirus software because of the observed reduced performance some platforms experience while running antivirus software. If clients turn on their antivirus software only initially, they fail to protect themselves during the most critical period—when they are actually connected to the network. Similarly, if a client remains logged in but never updates his or her antivirus software, the prophylactic advantage of scanning will severely diminish over time.

User-initiated disabling of antivirus software can be mitigated via a ticket-based extension to the proposed protocol. After successfully completing a set of test vectors, a client's platform could be issued a ticket as is done in authentication systems such as Kerberos [Steiner et al. 1998]. At a time close to the expiration of this ticket, the client would signal the server responsible for vetting clients to resend a new set of test vectors. Should a ticket expire, a client would be denied access to the network until such time when he or she could demonstrate the continued presence of properly functioning antivirus software.

The tests conducted during the benchmarking section of this paper give insight into the realistic granularity of ticket lifetimes. Given that it is possible to admit a machine into a network with nine-9s assurance in under half a second ($\bar{x}$ = 0.431 sec), requiring attached clients to reaffirm that they are running current antivirus software once every five minutes would amortize to approximately 0.144% of a machine's resources. Reaffirming the service once every minute would similarly require only 0.718% of the available system resources over time. Much like network time queries or auto-update functions, such assessment could be made totally transparent to the user. Depending on the user's tolerance for turning on and off his or her antivirus software, the vast majority of users would be persuaded to constantly run such a program, thereby helping to better secure the network.

The use of the ticket-based mechanism also increases the effort that must be expended to bypass this system. The nonmalicious adversary who is unwilling to install the requisite antivirus software would be forced to collect an enormous number of malware puzzles and their answers to *maintain* connectivity. The cost of performing inspection attacks on uninfected files, especially when expensive techniques such as static analysis are used, would also greatly increase. Assuming a sufficiently large database of malcode and a diverse set of methods for creating uninfected files, the cost associated with circumventing this protocol quickly eclipses the cost of legitimately interacting with the system.

An additional method of increasing the safety of the network would be to redirect clients failing the admission test to local repositories of antivirus updates. Such default behavior would allow users the opportunity to bring their platform up to a required safety specification and then join the network. Such a model would be particularly beneficial to systems such as hot spots that require users to log on before revenue can be collected.

5.4 Safety Issues

One of the largest issues facing the implementation and widespread distribution of this work is that of requiring average network administrators to maintain large databases of malcode. These repositories, while extremely valuable as a means of vetting protected users, could potentially be used as arsenals of weapons against known populations of unprepared machines. An adversary could request some number of puzzles and store all of the infected files received in the transaction. After amassing a sufficiently large cache of malware, the adversary could then attach to another network not implementing the same precautions and release its newly accumulated digital arsenal.

On reflection, however, we can see that this is not a serious threat. Dedicated adversaries do not need to use this system to obtain a cache of malcode. One could simply set up his or her own honeypot and catch a sufficiently large number of digital pathogens. This approach creates a potentially more dangerous stockpile, as many of the viruses and worms found by honeypots are previously unseen. An attacker could also employ the same method that we exploited—use search engines and track down Web sites where malcode is distributed. Because malcode is so widespread and so easy to locate and because we take great care in our design to prevent client infection, the means through which we certify clients is no more dangerous than allowing them to connect to the Internet.

If this is still considered problematic, we note that there are also alternatives to real viruses. The EICAR test virus [European Institute for Computer Anti-Virus Research 2003], for example, is a nonmalicious piece of code used to demonstrate the correct scanning ability of antivirus software without risking a real infection. The use of such test viruses has direct parallels to the use of vaccines in biological immune systems, in which case crippled or dead pathogens are introduced to the body so as to help build defenses without accidentally causing infection.

Very little research has been conducted into the creation of inoculated malcode. While some headway has been made in terms of creating realistic test viruses [Gordon 1995], many open problems remain. If, like in the world of immunology, the removal of the mechanism responsible for infection became simple, it seems likely that digital immunology could become a fruitful area of research. However, determining which sections of code can be successfully removed without affecting detection mechanisms is difficult. Much of the work done to combat polymorphic malcode [Fogla and Lee 2006; Kim and Karp 2004; Newsome et al. 2005; Singh et al. 2004] will likely be beneficial in this pursuit.

5.5 Generalized Certification

The preceding sections have focused on a demonstration of the certification procedure that measures the presence and correct operation of antivirus software. This technique, however, is not limited strictly to this application. For example, some networks perform a similar certification process by periodically scanning for open ports using the `nmap` utility [Insecure.org 2005].

One could also use our approach to vet implementations of essential algorithms: Determining that a machine is using robust implementations of cryptographic algorithms is possible by using a similar procedure. For example, a server could provide a client with a number of preimages with which to create keys, ciphertext, signatures, or HMACs. Signatures (in the noncryptographic sense) of known, weak implementations, such as poor sources of randomness, could be observed, depending on the client's response. Vulnerabilities in secure transmission suites such as SSH [OpenSSH] and a variety of VPN clients could also be discovered in a similar fashion. In networks where the transmission of highly sensitive data is critical, such proofs of robustness would be extremely valuable.

In essence, any program providing security services that provides a demonstrable result can be polled and fingerprinted through this technique. This work serves as a blueprint for implementing these tools. In the hopes of encouraging others to build similar infrastructure, we provide source code and documentation for our certification framework via http://siis.cse.psu.edu/tools/av-tools.html.

6. CONCLUSION

This work has considered the problem of noninvasive host certification. In so doing, we have asked, "How do we ensure that a host joining a network is following the proper security practices?" Determining whether a user or system is exercising appropriate security practices is difficult in any context. Commonly practiced techniques used to vet hosts, such as system scans, have the potential to infringe on user privacy and do not necessarily indicate the user's ability to self-protect. Other certification approaches, such as attestations, provide limited insight into software state—hence, they do not enable an appropriate level of certification of host configurations.

We have shown that it is possible for clients to prove the presence, proper functioning, and configuration of security infrastructure without allowing unrestricted access to their system. We apply this approach to certify that hosts are properly using up-to-date antivirus software. Users are given a vector of small files that may or may not contain malware. A host is certified if it can correctly identify the presence of malware in the test vectors. We have described our implementation and provide and demonstrate the feasibility of this work through performance analysis.

Our future work will seek to examine other means of employing this approach. In particular, we will explore how our zero-knowledge proof-inspired protocols can be used to certify more diverse security infrastructure. In the end, when combined with other techniques, such mechanisms may provide the kinds of certification desperately needed by contemporary networks.

REFERENCES

ARONSSON, H. A. 1995. Zero knowledge protocols and small systems. www.tml.hut.fi/Opinnot/Tik-110.501/1995/zeroknowledge.

BAILEY, M., COOKE, E., JAHANIAN, F., WATSON, D., AND NAZARIO, J. 2005. The blaster worm: Then and now. *IEEE Secur. Priv. Mag., 3*, 4, 26–31.

BELLOVIN, S. M. 1989. Security problems in the TCP/IP protocol suite. *SIGCOMM Comput. Comm. Rev. 19*, 2, 32–48.

BRICKELL, E., CAMENISCH, J., CHEN, L. 2004. Direct anonymous attestation. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*.

BULDAS, A., LAUD, P., AND LIPMAA, H. 2002. Eliminating counterevidence with applications to accountable certicate management. *J. Comput. Secur. 10*, 3, 273–296.

CHINCHANI, R. AND VAN DEN BERG, E. 2005. A fast static analysis approach to detect exploit code inside network flows. In *Proceedings of the International Symposium on Recent Advances in Intrusion Detection*.

COMPUTER EMERGENCY RESPONSE TEAM (CERT). www.cert.org.

CONGDON, P. 2003. RFC 3580 - IEEE 802.1X Remote authentication dial in user service (RADIUS) usage guidelines.

DIERKS, T. AND ALLEN, C. 1999. The TLS protocol version 1.0. *Internet Engineering Task Force*, RFC 2246.

EUROPEAN INSTITUTE FOR COMPUTER ANTI-VIRUS RESEARCH. 2003. Eicar—anti-virus test file. www.eicar.org/anti_virus_test_file.htm.

EUSTICE, K., KLEINROCK, L., MARKSTRUM, S., POPEK, G., RAMAKRISHNA, V., AND REIHER, P. 2003. Securing nomads: The case for quarantine, examination, and decontamination. In *Proceedings of the Workshop on New Security Paradigms*. 123–128.

EVERS, J. 2006. Microsoft's antivirus package makes a splash. http://news.com.com/2100-7355-6104926.html?tag=tb.

FOGLA, P. AND LEE, W. 2006. Evading network anomaly detection systems: Formal reasoning and practical techniques. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*.

GARETTO, M., GONG, W., AND TOWSLEY, D. 2003. Modeling malware spreading dynamics. In *Proceedings of IEEE INFOCOM*.

GARFINKEL, T., PFAFF, B., CHOW, J., ROSENBLUM, M., AND BONEH, D. 2003. Terra: A virtual machine-based platform for trusted computing. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP)*. 193–206.

GOLDWASSER, S., MICALI, S., AND RACKOFF, C. 1985. The knowledge complexity of interactive proof-systems. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*.

GORDON S. 1995. Is a good virus simulator still a bad idea? www.research.ibm.com/antivirus/SciPapers/Gordon/Simulators.html.

HARRIS, N. 2004. Securing network will help business owner mitigate legal liabilities. http://www.bizjournals.com/houston/stories/2004/01/19/focus5.html.

INSECURE.ORG. 2005. Nmap—Free security scanner for network exploration & security audits. www.insecure.org/nmap/.

INTEL CORPORATION. 2006. Intel 64 and IA-32 architectures; software developers manual, Volume 2A. http://www.intel.com/design/processor/manuals/253666.pdf.

JUELS, A. AND BRAINARD, J. 1999. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *Proceedings of the ISOC Network and Distributed System Security Symposium (NDSS)*.

KATO, T., TSUNEHIRO, T., TSUNODA, M., AND MIYAKE, J. 2003. A secure flash card solution for remote access for mobile workforce. *IEEE Trans. Consum. Electron. 49*, 561–566.

KAWASE, T., WATANABE, A., AND SASASE, I. 1998. Proposal of secure remote access using encryption. *Global Telecommunications Conference (GLOBECOM'98). The Bridge to Global Integration. IEEE, 2*, 868–873.

KIM, H. AND KARP, B. 2004. Autograph: Toward automated, distributed worm signature detection. In *USENIX Security Symposium*.

NEEDHAM, R. AND SCHROEDER, M. 1978. Using encryption for authentication in large networks of computers. *Commun. ACM, 21*, 993–999.

NEWSOME, J., KARP, B., AND SONG, D. 2005. Polygraph: Automatically generating signatures for polymorphic worms. In *IEEE Symposium on Security and Privacy*.

OLSEN, F. 2002. The growing vulnerability of campus networks. http://chronicle.com/free/v48/i27/27a03501.htm.

OPENSSH. www.openssh.com.

OPENSSL. www.openssl.org/.

PAXSON, V. 1999. Bro: A system for detecting network intruders in real-time. *Comput. Netw., 31*, 23–24.

RAHMAN, M. AND BHATTACHARYA, P. 2003. Remote access and networked appliance control using biometrics features. *IEEE Trans. Consum. Electron. 49*, 348–353.

SAILER, R., JAEGER, T., ZHANG, X., AND VAN DOORN, L. 2004. Attestation-based policy enforcement for remote access. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*.

SAILER, R., ZHANG, X., JAEGER, T., AND VAN DOORN, L. 2004. Design and implementation of a TCG-based integrity measurement architecture. In *Proceedings of the 13th USENIX Security Symposium*, 223–238.

SCHEUERMANN, D. 2002. The smartcard as a mobile security device. *Electron. Comm. Engin. J.* 205–210.

SINGH, S., ESTAN, C., VARGHESE, G., AND SAVAGE, S. 2004. Automated worm fingerprinting. In *ACM/USENIX Symposium on Operating System Design and Implementation (OSDI)*.

SNORT. The de facto standard for intrusion detection/prevention. www.snort.org.

STANIFORD, S., PAXSON, V., AND WEAVER, N. 2002. How to own the internet in your spare time. In *Proceedings of the USENIX Security Symposium*.

STEINER, J., NEUMAN, B., AND SCHILLER, J. 1998. Kerberos: An authentication service for open network systems. *USENIX*.

SYGATE WEB SITE. 2002. Sygate Secure Enterprise. www.sygate.com/products/sygate-secure-enterprise.htm.

SYMANTEC. Symantec Client Security. enterprisesecurity.symantec.com/products/products.cfm?ProductID=154.

TOTH, T. AND KRUEGEL, C. 2002. Accurate buffer overflow detection via abstract payload execution. In *Proceedings of the International Symposium on Recent Advances in Intrusion Detection*.

TRUSTED COMPUTING GROUP. www.trustedcomputinggroup.org.

WANG, K. AND STOLFO, S. J. 2004. Anomalous payload-based network intrusion detection. In *Proceedings of the International Symposium on Recent Advances in Intrusion Detection*.

WATERS, B., JUELS, A., HALDERMAN, J., AND FELTEN, E. 2004. New client puzzle outsourcing techniques for dos resistance. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*.

ZONE LABS. Zone Labs Integrity SecureClient. http://www.zonelabs.com/store/content/company/corpsales/secureClientOverview.jsp.

Received February 2007; revised August 2007; accepted September 2007