Counting in Regexes Considered Harmful:
Exposing ReDoS Vulnerability of Nonbacktracking Matchers

Lenka Turoňová Lukáš Holík Ivan Homoliak
Ondřej Lengál Tomáš Vojnar
Brno University of Technology, Czech Republic

Margus Veanes
Microsoft Research, USA

USENIX Security'22

Regular Expression Denial of Service (ReDoS)

What is a ReDoS? (in this talk)

- DoS by giving a regex matcher a hard input text

https://bit.ly/3uMlLsa

Regular Expression Denial of Service (ReDoS)

What is a ReDoS? (in this talk)

- DoS by giving a regex matcher a hard input text

`normal.txt` (500 kB):

```
abcdefghijklmopqrstuvwxyz01234
56789abcdefghijklmnopqrstuv...
```

```
$ time grep "a.{500}$" normal.txt
```

https://bit.ly/3uMlLsa

Regular Expression Denial of Service (ReDoS)

What is a ReDoS? (in this talk)

- DoS by giving a regex matcher a hard input text

`normal.txt` (500 kB):

```
abcdefghijklmopqrstuvwxyz01234
56789abcdefghijklmnopqrstuv...
```

```
$ time grep "a.{500}$" normal.txt
```

```
real 0.09
user 0.08
sys 0.00
```

https://bit.ly/3uMlLsa

Regular Expression Denial of Service (ReDoS)

What is a ReDoS? (in this talk)

- DoS by giving a regex matcher a hard input text

`normal.txt` (500 kB):

```
abcdefghijklmopqrstuvwxyz01234
56789abcdefghijklmnopqrstuv...
```

```
$ time grep "a.{500}$" normal.txt
```

```
real 0.09
user 0.08
sys 0.00
```

`evil.txt` (500 kB):

```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaa...
```

```
$ time grep "a.{500}$" evil.txt
```

https://bit.ly/3uMlLsa

Regular Expression Denial of Service (ReDoS)

What is a ReDoS? (in this talk)

- DoS by giving a regex matcher a hard input text

`normal.txt` (500 kB):

```
abcdefghijklmopqrstuvwxyz01234
56789abcdefghijklmnopqrstuv...
```

```
$ time grep "a.{500}$" normal.txt
```

```
real 0.09
user 0.08
sys 0.00
```

`evil.txt` (500 kB):

```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaa...
```

```
$ time grep "a.{500}$" evil.txt
```

```
real 1.63
user 1.52
sys 0.00
```

https://bit.ly/3uMlLsa

Regular Expression Denial of Service (ReDoS)

What is a ReDoS? (in this talk)

- DoS by giving a regex matcher a hard input text

`normal.txt` (500 kB):

```
abcdefghijklmopqrstuvwxyz01234
56789abcdefghijklmnopqrstuv...
```

```
$ time grep "a.{500}$" normal.txt
```

```
real 0.09
user 0.08
sys 0.00
```

`evil.txt` (500 kB):

```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaa...
```

```
$ time grep "a.{500}$" evil.txt
```

```
real 1.63
user 1.52
sys 0.00
```

19× slower!

(this may be a ReDoS)

(. . . and this is only a very simple example)

https://bit.ly/3uMlLsa

ReDoS

Real-world threat

- Stack Overflow, 2016: 34 minute outage (regex `"␣+$"`; line `"␣␣...␣a"` — $\overbrace{\text{␣␣...␣}}^{20{,}000\times}$)
- ReDoS vulnerability in Express.js (package `negotiator`), 2016
- ReDoS vulnerability in Node.js (package `url-regex`), 2020
- ...

Often caused by the use of backtracking matchers (PHP, JS, Perl, Ruby, .NET, ...)

https://bit.ly/3uMlLsa

ReDoS

Real-world threat

- Stack Overflow, 2016: 34 minute outage (regex `"␣+$"`; line `"`$\overbrace{\texttt{␣␣...␣}}^{20{,}000\times}$`a"`)
- ReDoS vulnerability in Express.js (package `negotiator`), 2016
- ReDoS vulnerability in Node.js (package `url-regex`), 2020
- . . .

Often caused by the use of backtracking matchers (PHP, JS, Perl, Ruby, .NET, . . .)

Solution: Use nonbacktracking matchers!

The Case of the Poisoned Event Handler: Weaknesses in the Node.js Event-Driven Architecture

James Davis, Virginia Tech, Blacksburg, VA, USA, davisjam@vt.edu
Gregor Kildow, Virginia Tech, Blacksburg, VA, USA, gregor@vt.edu
Dongyoon Lee, Virginia Tech, Blacksburg, VA, USA, dongyoon@vt.edu

A hybrid regex engine Except for back references, every feature of JavaScript regular expressions can be supported by a linear-time regular expression engine. As back

Freezing the Web: A Study of ReDoS Vulnerabilities in JavaScript-based Web Servers

Cristian-Alexandru Staicu, *Department of Computer Science, TU Darmstadt*
Michael Pradel, *Department of Computer Science, TU Darmstadt*

Another defense mechanism could be to use a more sophisticated regular expression engine that guarantees linear matching time. The problem is that these en-

Testing Regex Generalizability And Its Implications: A Large-Scale Many-Language Measurement Study

James C. Davis, Daniel Moyer, and Ayaan M. Kazerouni, Department of Computer Science, Virginia Tech, {davisjam, dmoyer, ayaan}@vt.edu
Dongyoon Lee, Department of Computer Science, Stony Brook University and Virginia Tech, dongyoon@cs.stonybrook.edu

assertions) in any programming language, Thompson's algorithm can be applied to almost all regexes in every programming language. This change would address most ReDoS vulnerabilities in a single stroke. We therefore urge program-

https://bit.ly/3uMlLsa

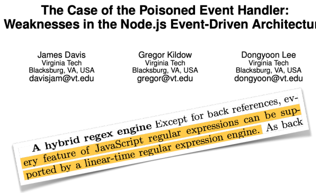
ReDoS

Real-world threat

- Stack Overflow, 2016: 34 minute outage (regex `"␣+$"`; line `"`$\overbrace{\texttt{␣␣...␣}}^{20{,}000\times}$`a"`)
- ReDoS vulnerability in Express.js (package `negotiator`), 2016
- ReDoS vulnerability in Node.js (package `url-regex`), 2020
- ...

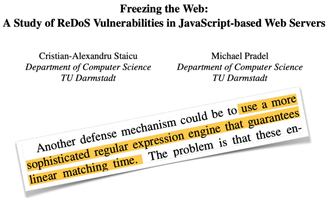
Often caused by the use of backtracking matchers (PHP, JS, Perl, Ruby, .NET, ...)

Solution: Use nonbacktracking matchers!


The Case of the Poisoned Event Handler: Weaknesses in the Node.js Event-Driven Architecture

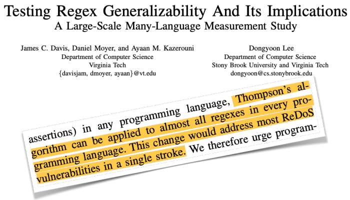
James Davis, Virginia Tech, Blacksburg, VA, USA, davisjam@vt.edu
Gregor Kildow, Virginia Tech, Blacksburg, VA, USA, gregor@vt.edu
Dongyoon Lee, Virginia Tech, Blacksburg, VA, USA, dongyoon@vt.edu

A hybrid regex engine Except for back references, every feature of JavaScript regular expressions can be supported by a linear-time regular expression engine. As back


Freezing the Web: A Study of ReDoS Vulnerabilities in JavaScript-based Web Servers

Cristian-Alexandru Staicu, *Department of Computer Science, TU Darmstadt*
Michael Pradel, *Department of Computer Science, TU Darmstadt*

Another defense mechanism could be to use a more sophisticated regular expression engine that guarantees linear matching time. The problem is that these en-


Testing Regex Generalizability And Its Implications: A Large-Scale Many-Language Measurement Study

James C. Davis, Daniel Moyer, and Ayaan M. Kazerouni, Department of Computer Science, Virginia Tech, {davisjam, dmoyer, ayaan}@vt.edu
Dongyoon Lee, Department of Computer Science, Stony Brook University and Virginia Tech, dongyoon@cs.stonybrook.edu

assertions) in any programming language, Thompson's algorithm can be applied to almost all regexes in every programming language. This change would address most ReDoS vulnerabilities in a single stroke. We therefore urge program-

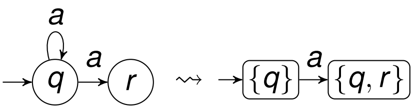
...or is it?

https://bit.ly/3uMlLsa

Nonbacktracking matchers

Nonbacktracking matchers:

- textbook: construct NFA, **determinize** ($\mathcal{O}(2^{|A|})$), perform match — linear time
  - $\rightsquigarrow$ DFA might be too big!! (often thousands, millions of macrostates)

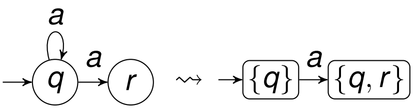


https://bit.ly/3uMlLsa

Nonbacktracking matchers

Nonbacktracking matchers:

- textbook: construct NFA, **determinize** ($\mathcal{O}(2^{|A|})$), perform match — linear time
  - $\rightsquigarrow$ DFA might be too big!! (often thousands, millions of macrostates)
- in practice (Thompson's algorithm):
  1. construct NFA
  2. determinize on-the-fly while doing membership test
  3. **cache!** $\rightsquigarrow$ $\mathcal{O}(|w|)$ average-case complexity



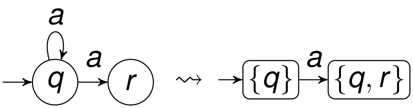
- tools: `grep`, `re2`, Rust, SRM, HYPERSCAN*

https://bit.ly/3uMlLsa

Nonbacktracking matchers

Nonbacktracking matchers:

- textbook: construct NFA, **determinize** ($\mathcal{O}(2^{|A|})$), perform match — linear time
  - $\rightsquigarrow$ DFA might be too big!! (often thousands, millions of macrostates)
- in practice (Thompson's algorithm):
  1. construct NFA
  2. determinize on-the-fly while doing membership test
  3. **cache!** $\rightsquigarrow$ $\mathcal{O}(|w|)$ average-case complexity



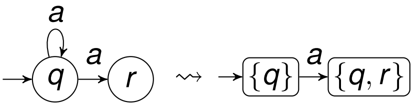
- tools: `grep`, `re2`, Rust, SRM, HYPERSCAN*
- can still run slow! due to low cache utilization
- How can we systematically generate ReDoS texts for nonbacktracking matchers?

https://bit.ly/3uMlLsa

Nonbacktracking matchers

Nonbacktracking matchers:

- textbook: construct NFA, **determinize** ($\mathcal{O}(2^{|A|})$), perform match — linear time
  - $\rightsquigarrow$ DFA might be too big!! (often thousands, millions of macrostates)
- in practice (Thompson's algorithm):
  1. construct NFA
  2. determinize on-the-fly while doing membership test
  3. **cache!** $\rightsquigarrow$ $\mathcal{O}(|w|)$ average-case complexity



- tools: `grep`, `re2`, Rust, SRM, HyperScan*
- can still run slow! due to low cache utilization
- How can we systematically generate ReDoS texts for nonbacktracking matchers?
- Exploit **counting**! (a.k.a. quantifiers, bounded repetition, . . .)

Counting in regexes

`a{5,42}` `b{100}`

https://bit.ly/3uMlLsa

How much are counting regexes prone to ReDoS?

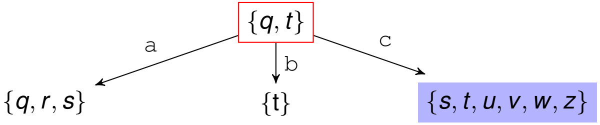
- 609,992 regexes (GitHub, SNORT, Bro, RegExLib, Microsoft, TrustPort, …)
- removed unsupported (look-arounds/back-references/…)
- $\rightsquigarrow$ 443,265 regexes
- classify according to sum of upper bounds in counting, e.g., `a{5, 42 }`
- DFA Big: ≥1,000 states (often the size of DFA cache)

regex set	**#**	**#DFA big**	**%**
no counting	395,752	175	0.04 %
counting bounds ≤20	39,414	343	0.8 %
counting bounds >20	8,099	1,600	20. %

https://bit.ly/3uMlLsa

ReDoS generator for nonbacktracking matchers

- generate input text by search through the DFA
  - generate non-matching text
  - prefer macrostates that are
    1. unvisited (matcher cache miss)
    2. big (hard to compute successors)

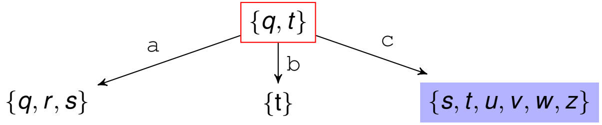


  - $\rightsquigarrow$ try to enforce $\mathcal{O}(|w| \cdot |A|)$ runtime ($A =$ the NFA; $|A| = |Q| + |\Delta|$)

https://bit.ly/3uMlLsa

ReDoS generator for nonbacktracking matchers

- generate input text by search through the DFA
  - generate non-matching text
  - prefer macrostates that are
    1. unvisited (matcher cache miss)
    2. big (hard to compute successors)

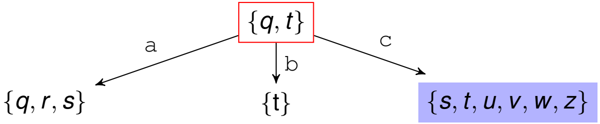


  - $\rightsquigarrow$ try to enforce $\mathcal{O}(|w| \cdot |A|)$ runtime $\quad$ (A = the NFA; $|A| = |Q| + |\Delta|$)
- **Issue**: how to navigate to big macrostates?
  - DFA too big, cannot construct!
  - $\rightsquigarrow$ instead of DFA, use **Counting-Set Automaton** [OOPSLA'20]
    - allows compact deterministic representation of regexes with counting

https://bit.ly/3uMlLsa

ReDoS generator for nonbacktracking matchers

- generate input text by search through the DFA
  - generate non-matching text
  - prefer macrostates that are
    1. unvisited (matcher cache miss)
    2. big (hard to compute successors)



  - $\rightsquigarrow$ try to enforce $\mathcal{O}(|w| \cdot |A|)$ runtime ($A =$ the NFA; $|A| = |Q| + |\Delta|$)
- **Issue**: how to navigate to big macrostates?
  - DFA too big, cannot construct!
  - $\rightsquigarrow$ instead of DFA, use **Counting-Set Automaton** [OOPSLA'20]
    - allows compact deterministic representation of regexes with counting
- ReDoS generator `GadgetCA`

https://bit.ly/3uMlLsa

Experiments

Dataset:

- 609,992 regexes (GitHub, SNORT, Bro, RegExLib, Microsoft, TrustPort, . . .)
- removed unsupported (look-arounds/back-references/. . .)
- keep regexes with sum of counter bounds $>$20
- $\rightsquigarrow$ 8,099 regexes

https://bit.ly/3uMlLsa

Experiments

Dataset:

- 609,992 regexes (GitHub, SNORT, Bro, RegExLib, Microsoft, TrustPort, . . .)
- removed unsupported (look-arounds/back-references/. . .)
- keep regexes with sum of counter bounds $>$20
- $\rightsquigarrow$ 8,099 regexes

Other generators:

- `RXXR2`, `RegexCheck`, `RegexStatic`, `Rescue`
- they target backtracking matchers

https://bit.ly/3uMlLsa

Experiments

Dataset:

- 609,992 regexes (GitHub, Snort, Bro, RegExLib, Microsoft, TrustPort, . . .)
- removed unsupported (look-arounds/back-references/. . .)
- keep regexes with sum of counter bounds $>$20
- $\rightsquigarrow$ 8,099 regexes

Other generators:

- `RXXR2`, `RegexCheck`, `RegexStatic`, `Rescue`
- they target backtracking matchers

$\sim$50 MB input text from each generator for each regex and try on different matchers

https://bit.ly/3uMlLsa

How many ReDoSes could we generate?

- ReDoS: time $>100\times$ longer than average for matcher on random input
  - results for other ReDoS criteria in the paper
- **`GadgetCA`**: different strategies for exploring the counting-set automaton
  - **OneLine**: special strategy to target HyperScan

Generators		$>100\times$ **AVG$_{REGEX}$-ReDoS attacks** (8,099 regexes)												
		grep	**re2**	**rust**	**srm**	**hyper-scan**	**ca**	**ruby**	**php**	**perl**	**python**	**java**	**java-Script**	**.NET**
`GadgetCA`	**Counting**	1157	1465	1066	279	2	3	1085	796	1252	407	142	140	171
`GadgetCA`	**OneLine**	966	15	57	16	23	0	199	9	208	277	232	228	238
`GadgetCA`	**Greedy**	878	14	57	12	0	0	164	9	174	232	190	194	203
`GadgetCA`	**Random**	1066	320	292	130	0	0	153	156	266	91	63	60	72
`RXXR2`		1	0	2	0	0	0	10	0	4	22	8	8	20
`RegexCheck`		4	0	4	0	0	0	3	0	0	4	3	2	2
`RegexStatic`		47	5	5	0	0	0	80	14	49	137	125	134	90
`Rescue`		1	2	4	0	0	1	12	2	6	15	7	6	14
		nonbacktracking						**backtracking**						

(**ca**: our matcher based on counting-set automata)

https://bit.ly/3uMlLsa

Real-world security solutions

Real-life SNORT rule-sets (Emerging Threats Pro and 3CORESec, Talos)

- filtered out unsupported regexes and those with the sum of repetition bounds ≤ 20
- obtained 1,112 regexes (from 22,425)
- slowdown of **evil** vs. **random** text

https://bit.ly/3uMlLsa

Real-world security solutions

Real-life SNORT rule-sets (Emerging Threats Pro and 3CORESec, Talos)

- filtered out unsupported regexes and those with the sum of repetition bounds ≤ 20
- obtained 1,112 regexes (from 22,425)
- slowdown of **evil** vs. **random** text

SNORT3@HYPERSCAN

- HYPERSCAN: no cache (modified alg.)
- TCP reassembly off
- MTU 1.5 kB and 9 kB in 100 MB files

https://bit.ly/3uMlLsa

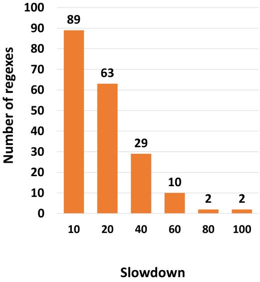
Real-world security solutions

Real-life SNORT rule-sets (Emerging Threats Pro and 3CORESec, Talos)

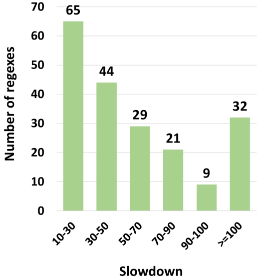
- filtered out unsupported regexes and those with the sum of repetition bounds ≤ 20
- obtained 1,112 regexes (from 22,425)
- slowdown of **evil** vs. **random** text

SNORT3@HYPERSCAN

- HYPERSCAN: no cache (modified alg.)
- TCP reassembly off
- MTU 1.5 kB and 9 kB in 100 MB files



SNORT3@HYPERSCAN (1.5 kB)



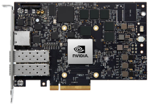
SNORT3@HYPERSCAN (9 kB)

examples	**1.5 kB**	**9 kB**
`"[?&]u=[^&\s]{35}"`	79×	214×
`"src\s*\x3D(3D)?\s*['"][^'"]{244}"`	71×	164×
`"[?&](cmd|pwd|usr)=[^&]{64}"`	43×	108×

https://bit.ly/3uMlLsa

NVIDIA BlueField-2

- DPU: data processing unit (ASIC)
- 2×25 GbE interfaces, 8 ARMs
- HW-accelerated regex matching unit: ∼40 Gbps
- 100 GB files (continuous)
- 617 regexes (from 1,112; unsupported: 495)

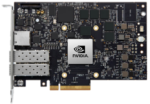


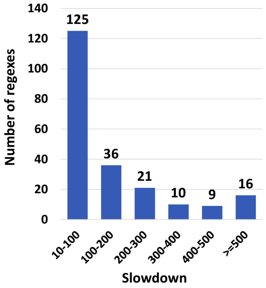
https://bit.ly/3uMlLsa

NVIDIA BlueField-2

NVIDIA BlueField-2

- DPU: data processing unit (ASIC)
- 2×25 GbE interfaces, 8 ARMs
- HW-accelerated regex matching unit: ∼40 Gbps
- 100 GB files (continuous)
- 617 regexes (from 1,112; unsupported: 495)





NVIDIA BlueField-2

examples	slowdown
"\sPARTIAL.*BODY\.PEEK\[[^\]]{1024}"	2,194×
"\s{230,}\.htr"	956×
"object\s[^>]*type\s*=\s*[\x22\x27][^\x22\x27]*\x2f{32}"	655×

https://bit.ly/3uMlLsa

Conclusion

- nonbacktracking regex matchers are **NOT** a silver bullet against ReDoS
- they can still be slowed down, often by attacking counting, e.g., `a{100}`
- generator that can exploit counting — **`GadgetCA`**

https://bit.ly/3uMlLsa

Conclusion

- nonbacktracking regex matchers are **NOT** a silver bullet against ReDoS
- they can still be slowed down, often by attacking counting, e.g., `a{100}`
- generator that can exploit counting — **`GadgetCA`**
- mitigation:
  - obvious ones (time limit, input limit, disallow counting)
  - overapproximate: `a{5,42}` ⇝ `a*`
  - detect vulnerable regexes with **`GadgetCA`**
  - use better regex matching technology (e.g., counting-set automata)

https://bit.ly/3uMlLsa

Conclusion

- nonbacktracking regex matchers are **NOT** a silver bullet against ReDoS
- they can still be slowed down, often by attacking counting, e.g., `a{100}`
- generator that can exploit counting — **`GadgetCA`**
- mitigation:
  - obvious ones (time limit, input limit, disallow counting)
  - overapproximate: `a{5,42}` $\rightsquigarrow$ `a*`
  - detect vulnerable regexes with **`GadgetCA`**
  - use better regex matching technology (e.g., counting-set automata)

Thank you!

https://bit.ly/3uMlLsa

Appendix

https://bit.ly/3uMlLsa

Generators		>100×$AVG_{MATCHER}$-ReDoS attacks												
		grep	re2	rust	srm	hyper-scan	ca	ruby	php	perl	python	java	java-Script	.NET
GadgetCA	GREEDY	1741	15	95	18	2	40	260	38	382	367	328	314	431
GadgetCA	COUNTING	2457	742	1016	300	5	67	1355	1596	1473	277	279	258	416
GadgetCA	RANDOM	2033	120	122	289	3	46	348	388	412	176	177	117	258
GadgetCA	ONELINE	1796	17	99	23	20	53	322	34	441	448	405	379	521
RXXR2		13	0	2	0	0	1	24	0	5	30	10	10	34
RegexCheck		104	0	5	0	1	0	7	1	7	11	8	4	14
RegexStatic		93	1	9	0	1	7	159	50	80	263	253	243	279
Rescue		12	0	3	0	0	2	23	2	5	23	13	12	26

https://bit.ly/3uMlLsa

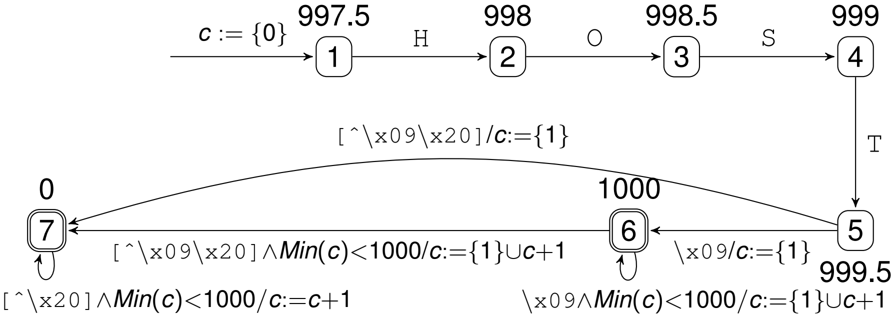
Generators		>100s-ReDoS attacks												
		grep	re2	rust	srm	hyper-scan	ca	ruby	php	perl	python	java	java-Script	.NET
GadgetCA	GREEDY	192	72	76	238	0	61	1087	1408	56	200	215	210	390
GadgetCA	COUNTING	216	110	96	272	0	45	1724	1979	89	218	242	211	419
GadgetCA	RANDOM	126	28	48	123	0	46	682	885	60	160	181	111	334
GadgetCA	ONELINE	192	17	32	23	0	56	333	40	187	433	414	378	584
RXXR2		7	0	2	0	0	1	24	0	4	30	11	11	34
RegexCheck		14	0	2	0	0	0	7	1	1	9	8	4	16
RegexStatic		34	1	5	0	0	8	160	63	69	262	253	243	285
Rescue		12	0	3	0	0	2	23	3	4	23	13	12	27
random text		52	4	11	17	0	82	33	47	23	109	162	36	231

https://bit.ly/3uMlLsa

Generators		>10s-ReDoS attacks												
		grep	re2	rust	srm	hyper-scan	ca	ruby	php	perl	python	java	java-Script	.NET
GadgetCA	GREEDY	1058	703	274	311	1	135	5050	6580	837	1027	485	955	2629
	COUNTING	1181	1116	295	391	3	121	5440	6289	1294	1503	532	1317	3000
	RANDOM	713	135	259	242	1	106	4405	5389	361	523	385	410	2025
	ONELINE	576	17	78	30	6	130	540	69	402	678	637	485	1448
RXXR2		11	0	2	0	0	1	26	0	5	33	12	13	35
RegexCheck		25	0	3	0	1	0	7	3	7	18	15	9	36
RegexStatic		78	1	9	0	0	19	182	70	78	287	274	254	333
Rescue		11	0	3	0	0	4	24	2	5	26	13	13	28
random text		153	10	70	27	2	137	175	47	147	272	255	228	698

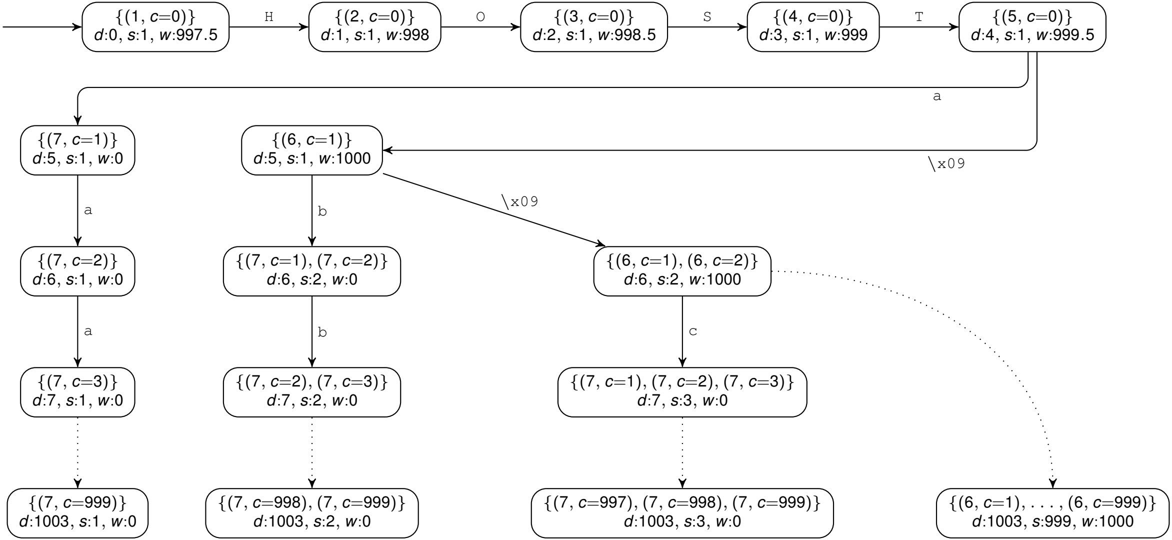
https://bit.ly/3uMlLsa

Counting-set automaton with weights



CSA with weights for the regex "^HOST\x09*[^\x20]{1000}"

https://bit.ly/3uMlLsa



DFA states explored by our algorithm on the regex `"^HOST\x09*[^\x20]{1000}"`

`https://bit.ly/3uMlLsa`