ReDoSHunter: A Combined Static and Dynamic Approach for Regular Expression DoS Detection

Yeting Li, Zixuan Chen, Jialun Cao, Zhiwu Xu, Qiancheng Peng,
Haiming Chen, Liyuan Chen, Shing-Chi Cheung

Institute of Software, Chinese Academy of Sciences
University of Chinese Academy of Sciences
Shenzhen University
The Hong Kong University of Science and Technology
Tencent

Regular Expression Denial-of-Service

The Server Side

Validate the input email using a vulnerable regex:

^[a-zA-Z0-9._]+@([a-zA-Z0-9]+.)+com$

Enter your email:

The Client Side

Normal input :
abc@def.com

Respond immediately

Malicious input :
c@ccccccccc...

Denial-of-Service

Regular Expression Denial-of-Service

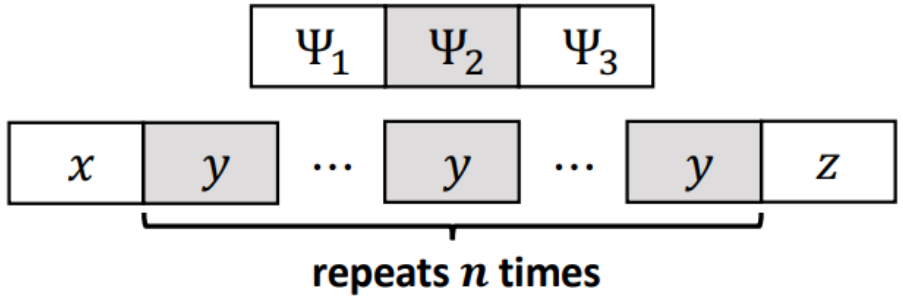
- Existing Solutions
  - Static approaches : low precision
  - Dynamic approaches : low recall

- Our Solution
  - ReDoSHunter: a combined static and dynamic approach for ReDoS detection
  - Propose five patterns of ReDoS-vulnerable regexes

Components of Attack String

ReDoS-vulnerable Regex $\mathbf{r} = \Psi_1\Psi_2\Psi_3$

Construct Attack String $w = xy^nz\ (n > 0)$
where $x \in \mathcal{L}(\Psi_1)$, $y^n \in \mathcal{L}(\Psi_2)$ **and** $w \notin \mathcal{L}(\mathbf{r})$



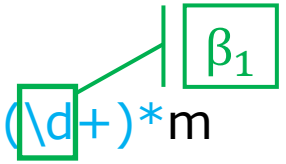
- For example, r_1 = ^[a-zA-Z0-9._]+@([a-zA-Z0-9]+.)+com$

Ψ_1 = [a-zA-Z0-9._]@	Ψ_2 = ([a-zA-Z0-9]+.)+	Ψ_3 = com
x = 'c@' $\in \mathcal{L}(\Psi_1)$	y = 'c' $\in \mathcal{L}(\Psi_2)$	z = '!' $\notin \mathcal{L}(\Psi_3)$

$w = xy^nz$ = 'c@' + 'c' × n + '!'

Five ReDoS Patterns

- Nested Quantifiers (NQ)
  - Optional nested quantifiers result in two choices for each pump strings
  - Vuln. type: Exponential
  - For example, CVE-2015-9239, regex: \[(\d+;)?(\d+)*m



attack string: ‘[’ + ‘0’ ×30 + ‘!’

Five ReDoS Patterns

- Exponential Overlapping Disjunction (EOD)
  - A disjunction with a common outer quantifier whose multiple nodes overlap
  - Formally, an EOD is of the form $\beta = (\cdots(\beta_1|\beta_2|\cdots|\beta_k)\cdots)\{m_\beta, n_\beta\}$ with $n_\beta > 1$, satisfying one of the following conditions

No.	Condition
#1	$\beta_p.\text{first} \cap \beta_q.\text{first} \neq \varnothing$, where $1 \leq p,q \leq k$ and $p \neq q$
#2	$\beta_p.\text{first} \cap \beta_q.\text{followlast} \neq \varnothing$, where $1 \leq p,q \leq k$ and $p \neq q$

The head of β_p and the head of β_q overlap

The head of β_p and the tail of β_q overlap

  - Vuln. type: Exponential
  - For example, CVE-2020-7662,

regex: "(?:\\[\x00-\x7f]|[^\x00-\x08\x0a-\x1f\x7f"])*)"

(β_1 = \\[\x00-\x7f], β_2 = [^\x00-\x08\x0a-\x1f\x7f"])

attack string: '"' + '\\\x7e' ×30 + '!'

Five ReDoS Patterns

- Exponential Overlapping Adjacency (EOA)
  - Two overlapping nodes with a common outer quantifier {m,n} (n>1) are either adjacent or can reach each other by skipping some optional nodes
  - Formally, an EOA is of the form $\beta = (\cdots(\beta_1\beta_2)\cdots)\{m_\beta, n_\beta\}$ with $n_\beta > 1$, satisfying one of the following conditions

No.	Condition
#1	$(\beta_1.\text{followlast} \cup \beta_1.\text{last}) \cap \beta_2.\text{first} \neq \varnothing$
#2	$\beta_1.\text{first} \cap (\beta_2.\text{followlast} \cup \beta_2.\text{last}) \neq \varnothing$

The tail of β_1 and the head of β_2 overlap

The head of β_1 and the tail of β_2 overlap

  - Vuln. type: Exponential
  - For example, CVE-2018-3738, regex:^(?:\.?[a-zA-Z_]β_1[a-zA-Z_0-9]*β_2)+$

    attack string: ‘a’ ×30 + ‘!’

Five ReDoS Patterns

- Polynomial Overlapping Adjacency (POA)
  - Two overlapping nodes with an optional common outer quantifier {0,1} are either adjacent or can reach each other by skipping some optional nodes
  - Formally, an POA is of the form $\beta = (\cdots(\beta_1\beta_2)\cdots)\{m_\beta, n_\beta\}$ with $n_\beta \leq 1$, satisfying the following conditions

No.	Condition
#1	$\beta_1.\text{followlast} \cap \beta_2.\text{first} \neq \varnothing$

The tail of β_1 and the head of β_2 overlap

  - Vuln. type: Polynomial
  - For example, CVE-2018-3737,

regex: ^([a-z0-9-]+)[\t]+([a-zA-Z0-9+\/]+[=]*)([\n \t]+([^\n]+))?$

β_1 β_2

attack string: '0\t0' + '\t' × 10000 + '\n'

Five ReDoS Patterns

- Starting with Large Quantifier (SLQ)
  - The regex engine keeps moving the regex starting with a large quantifier across the string to find a match
  - Formally, an SLQ satisfying one of the right conditions, where $n_\beta \geq n_{min}$, n_{min} is a pre-defined number for the minimal repetitions

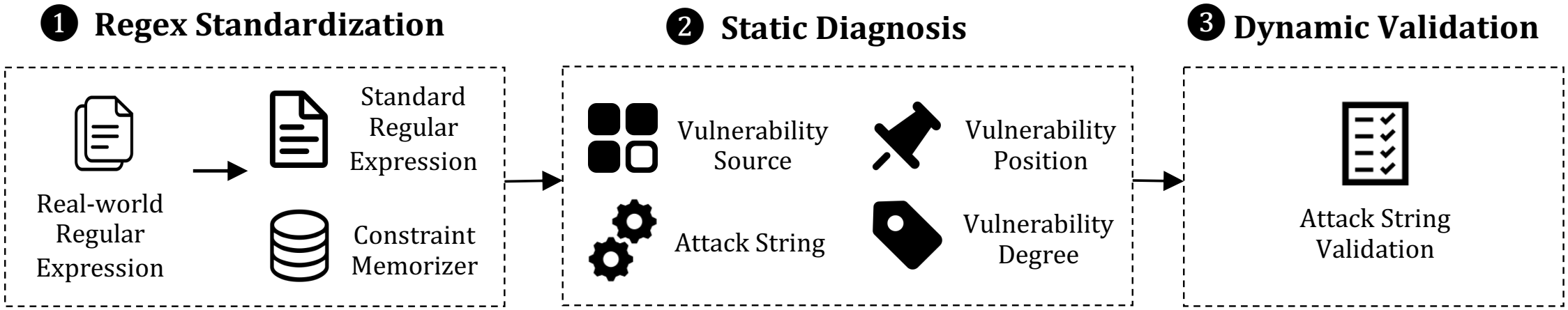
No.	Condition
#1	starting with $\beta_1\{m_\beta, n_\beta\}$
#2	starting with $\beta_1\beta_2\{m_\beta, n_\beta\}$ such that $(\mathcal{L}(\beta_1) \cap \mathcal{L}(\beta_2\{m_\beta, n_\beta\}))\backslash\{\varepsilon\} \neq \varnothing$
#3	starting with $\beta_1(\gamma_1\|\gamma_2\|\ldots\|\gamma_k)\{m_\beta, n_\beta\}$ such that there exists a word $w = w_0w_1\ldots w_\ell \in \mathcal{L}(\gamma_p\{m_\beta, n_\beta\})$, $w_1\ldots w_\ell w_0 \in \mathcal{L}(\gamma_q\{m_\beta, n_\beta\})$, and $w_0 \in \mathcal{L}(\beta_1)$
#4	starting with $\beta_1(\gamma_1\gamma_2\ldots\gamma_k)\{m_\beta, n_\beta\}$ such that all the $\gamma_1, \gamma_2, \ldots, \gamma_k$ are nullable, and there exists a word $w = w_0w_1\ldots w_\ell \in \mathcal{L}(\gamma_p\{m_\beta, n_\beta\})$, $w_1\ldots w_\ell w_0 \in \mathcal{L}(\gamma_q\{m_\beta, n_\beta\})$, and $w_0 \in \mathcal{L}(\beta_1)$

  - Vuln. type: Polynomial
  - For example, CVE-2019-1010266,

    regex:[a-z][A-Z]|[A-Z]{2,}[a-z]|[0-9][a-zA-Z]|[a-zA-Z][0-9]|[^a-zA-Z0-9]

attack string: ‘C’ × 10000 + ‘!’

ReDoSHunter Overview



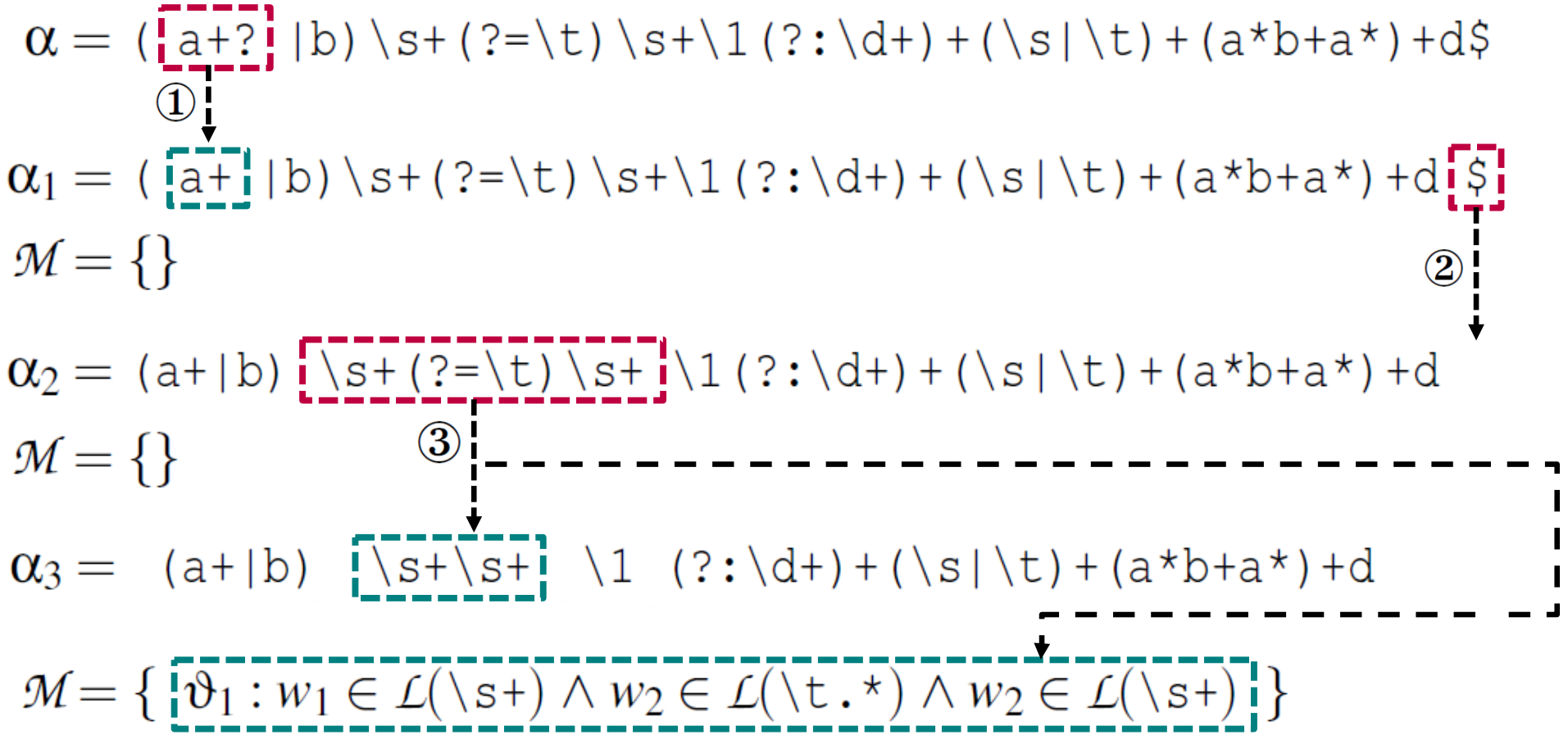
Regex Standardization

- Real-world regex → Standard regex + Constraints
  - We first convert a given regex into a standard regex with some constraints
  - Not to give an equivalent transformation, but instead trying to give a transformation with the same effect on ReDoS
  - The source regex has ReDoS-vulnerabilities iff the transformed target regex has the same ReDoS-vulnerabilities

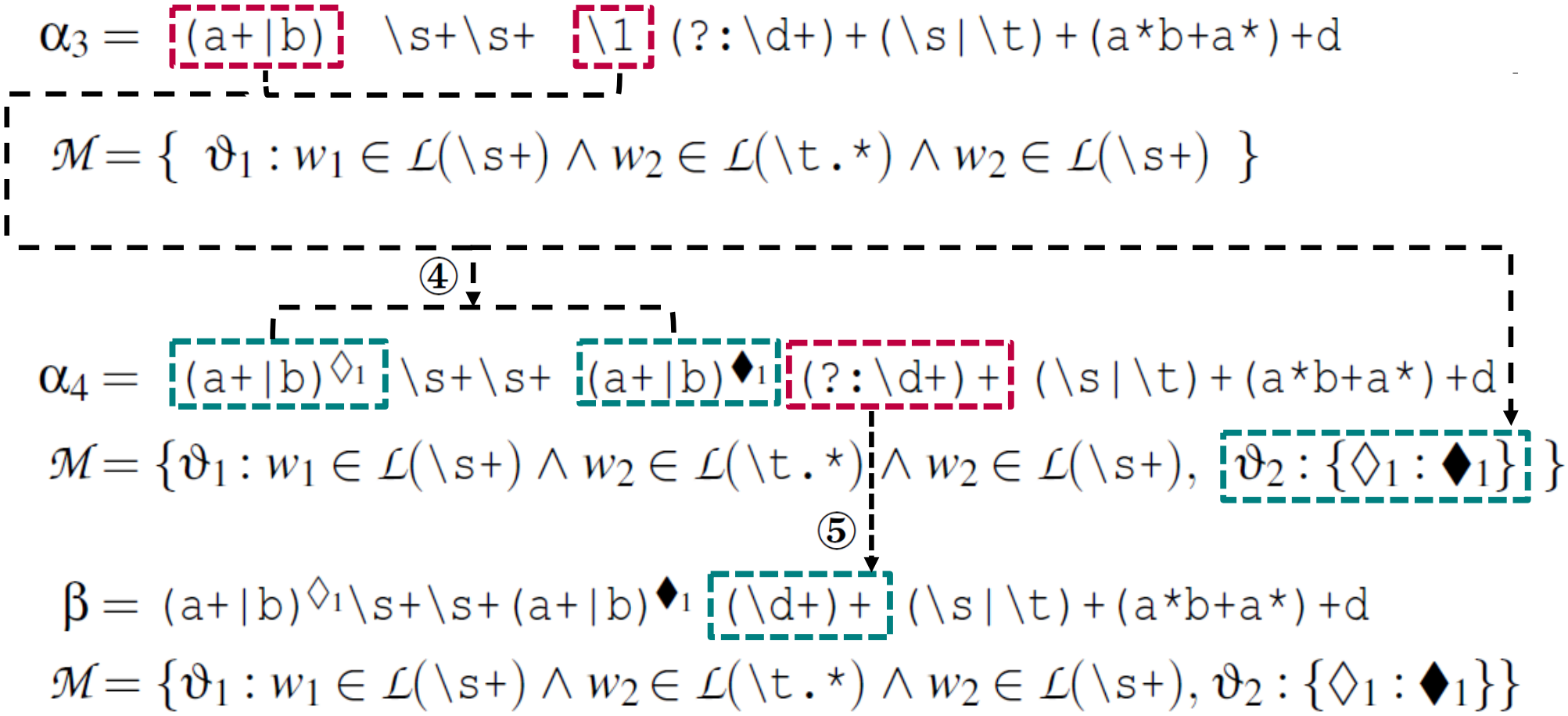
Next, We use a regex α to illustrate our algorithms,

$$\alpha = (a+?|b)\s+(?=\t)\s+\1(?:\d+)+(\s|\t)+(a*b+a*)+d\$$$

Regex Standardization



Regex Standardization



Static Diagnosis

$$\mathcal{M} = \{\vartheta_1 : w_1 \in \mathcal{L}(\backslash s+) \wedge w_2 \in \mathcal{L}(\backslash t.*) \wedge w_2 \in \mathcal{L}(\backslash s+), \vartheta_2 : \{\Diamond_1 : \blacklozenge_1\}\}$$

$$\beta = (a+|b)^{\Diamond_1}\backslash s+\backslash s+(a+|b)^{\blacklozenge_1}(\backslash d+)+(\backslash s|\backslash t)+(a*b+a*)+d$$

SLQ:
prefix string $x = \varepsilon$
infix string $y =$ 'a'
suffix string $z =$ '!'

NQ:
prefix string $x =$ 'a\n\ta'
infix string $y =$ '1'
suffix string $z =$ '!'

EOA:
prefix string $x =$ 'a\n\ta1\t'
infix string $y =$ 'ba'
suffix string $z =$ '!'

POA:
prefix string $x =$ 'a'
infix string $y =$ '\t\t'
suffix string $z =$ '!'

EOD:
prefix string $x =$ 'a\n\ta1'
infix string $y =$ '\t'
suffix string $z =$ '!'

Static Diagnosis

$$\alpha = (a+?|b)\backslash s+(?=\backslash t)\backslash s+\backslash 1(?:\backslash d+)+(\backslash s|\backslash t)+(a*b+a*)+d\$$$

No.	Pattern	Vuln. Degree	Vuln. Position	Attack String
#1	NQ	Exponential	(?:\d+)+	‘a\n\ta’ + ‘1’ × 30 + ‘!’
#2	EOD	Exponential	(\s|\t)+	‘a\n\ta1’ + ‘\t’ × 30 + ‘!’
#3	EOA	Exponential	(a*b+a*)+	‘a\n\ta1\t’ + ‘ba’ × 30 + ‘!’
#4	POA	Polynomial	\s+(?=\t)\s+	‘a’ + ‘\t\t’ × 10000 + ‘!’
#5	SLQ	Polynomial	a+?	‘a’ × 10000 + ‘!’

Dynamic Validation

Step1. Measure the time t for the source regex α to match the attack string w

Step2. Check whether the corresponding threshold T_P (for polynomial vulnerability) or T_E (for exponential vulnerability) is triggered

$$\alpha = (a+?|b)\backslash s+(?=\backslash t)\backslash s+\backslash 1(?:\backslash d+)+(\backslash s|\backslash t)+(a*b+a*)+d\$$$

No.	Pattern	Vuln. Degree	Vuln. Position	Attack String	Validated
#1	NQ	Exponential	(?:\d+)+	‘a\n\ta’ + ‘1’ × 30 + ‘!’	√
#2	EOD	Exponential	(\s|\t)+	‘a\n\ta1’ + ‘\t’ × 30 + ‘!’	√
#3	EOA	Exponential	(a*b+a*)+	‘a\n\ta1\t’ + ‘ba’ × 30 + ‘!’	√
#4	POA	Polynomial	\s+(?=\t)\s+	‘a’ + ‘\t\t’ × 10000 + ‘!’	√
#5	SLQ	Polynomial	a+?	‘a’ × 10000 + ‘!’	√

Experiment Setup

- Benchmark Datasets
  - Regex sets: 37,651 regexes from Corpus, RegExLib, and Snort
  - Known ReDoS-vulnerabilities: 35 CVEs with clear descriptions or sources
  - Intensively-tested projects: 26 popular projects on GitHub/npm/PyPI

Table 7: The Regex Sets for Evaluation.

Name	Number	Avg Len	Description
Corpus	13,597	33.97	Regexes from scraped Python projects
RegExLib	8,699	69.75	Online regex examples from regexlib.com
Snort	15,355	92.28	Regexes extracted from the Snort NIDS for inspecting IP packets
Total:	37,651		

- Baselines
  - Static approaches: RXXR2, Rexploiter, NFAA, safe-regex and Regexploit
  - Dynamic approaches: SDL and ReScue

Experiment Setup

- Evaluation Metrics
  - Precision: the proportion of the true positives over the reported vulnerabilities.

    Prec = TPs / (TPs + FPs)
  - Recall: the proportion of the true positive over all the real vulnerabilities.

    Rec = TPs / (TPs + FNs)

- Configuration
  - $N_P = 30{,}000$, $N_E = 100$, $T_P = 1s$, $T_E = 0.1s$, and $n_{min} = 100$

Results on Regex Benchmarks

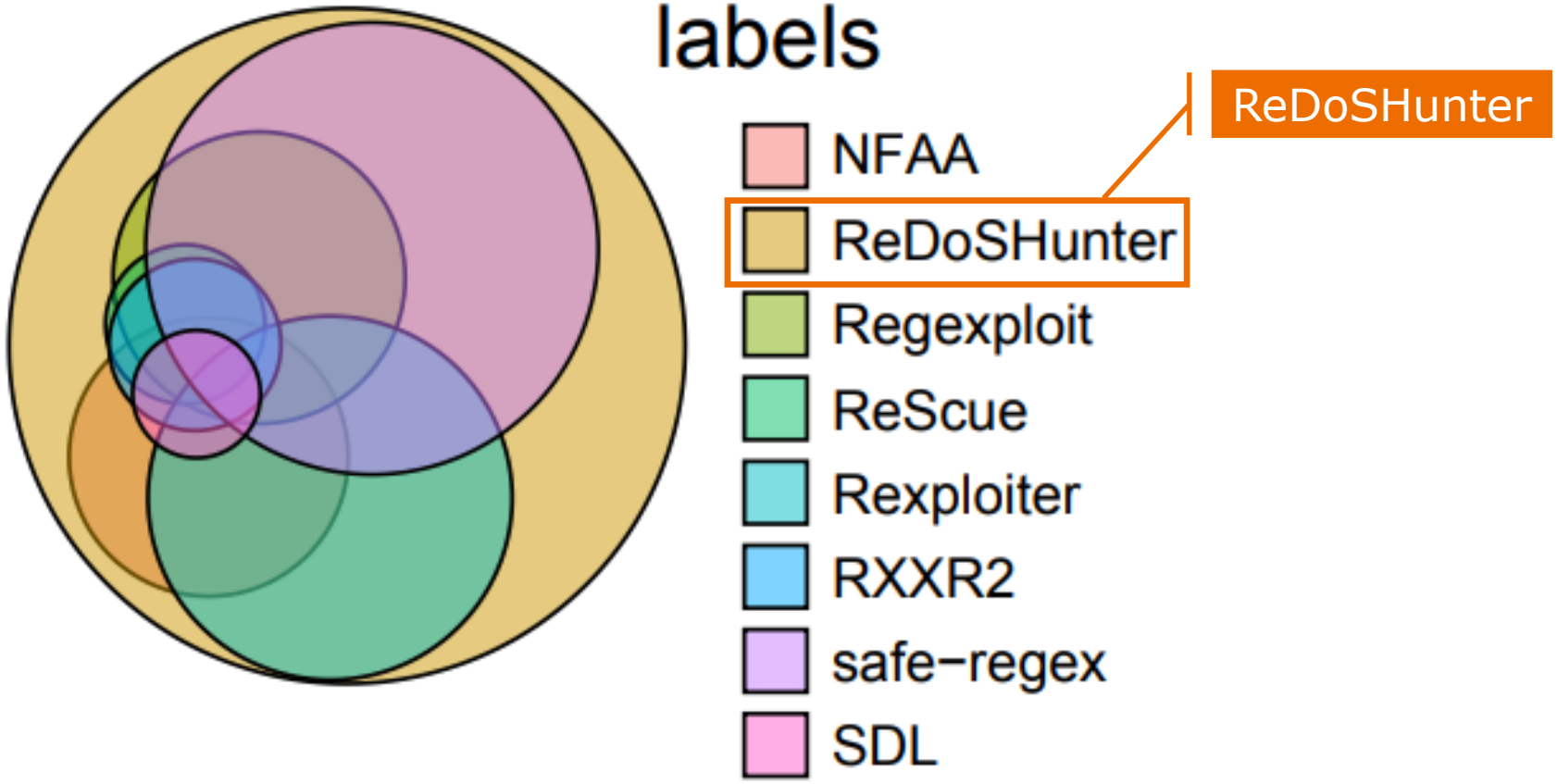
Static approaches

Regex Engine	Java-8					Java-13					Python-3.7					Node.js-14				
Technique	TP	FP	FN	Prec (%)	Rec (%)	TP	FP	FN	Prec (%)	Rec (%)	TP	FP	FN	Prec (%)	Rec (%)	TP	FP	FN	Prec (%)	Rec (%)
RXXR2	224	5	10,121	97.82	2.17	216	13	10,032	94.32	2.11	213	16	9,594	93.01	2.17	219	10	9,427	95.63	2.27
Rexploiter	2,052	288	8,293	87.69	19.84	2,041	299	8,207	87.22	19.92	1,955	385	7,852	83.55	19.93	1,915	425	7,731	81.84	19.85
NFAA	975	13	9,370	98.68	9.42	968	20	9,280	97.98	9.45	857	131	8,950	86.74	8.74	842	146	8,804	85.22	8.73
safe-regex	3,760	2,348	6,585	61.56	36.35	3,715	2,393	6,533	60.82	36.25	3,586	2,522	6,221	58.71	36.57	3,540	2,568	6,106	57.96	36.70
Regexploit	1,051	2	9,294	99.81	10.16	1,051	2	9,197	99.81	10.26	1,044	9	8,763	99.15	10.65	1,032	21	8,614	98.01	10.70
SDL	112	0	10,233	100	1.08	108	4	10,140	96.43	1.05	98	14	9,709	87.50	1.00	102	10	9,544	91.07	1.06
ReScue	188	0	10,157	100	1.82	183	5	10,065	97.34	1.79	175	13	9,632	93.09	1.78	179	9	9,467	95.21	1.86
ReDoSHunter	**10,345**	**0**	**0**	**100**	**100**	**10,248**	**0**	**0**	**100**	**100**	**9,807**	**0**	**0**	**100**	**100**	**9,646**	**0**	**0**	**100**	**100**
Real Vulnerabilities	10,345					10,248					9,807					9,646				

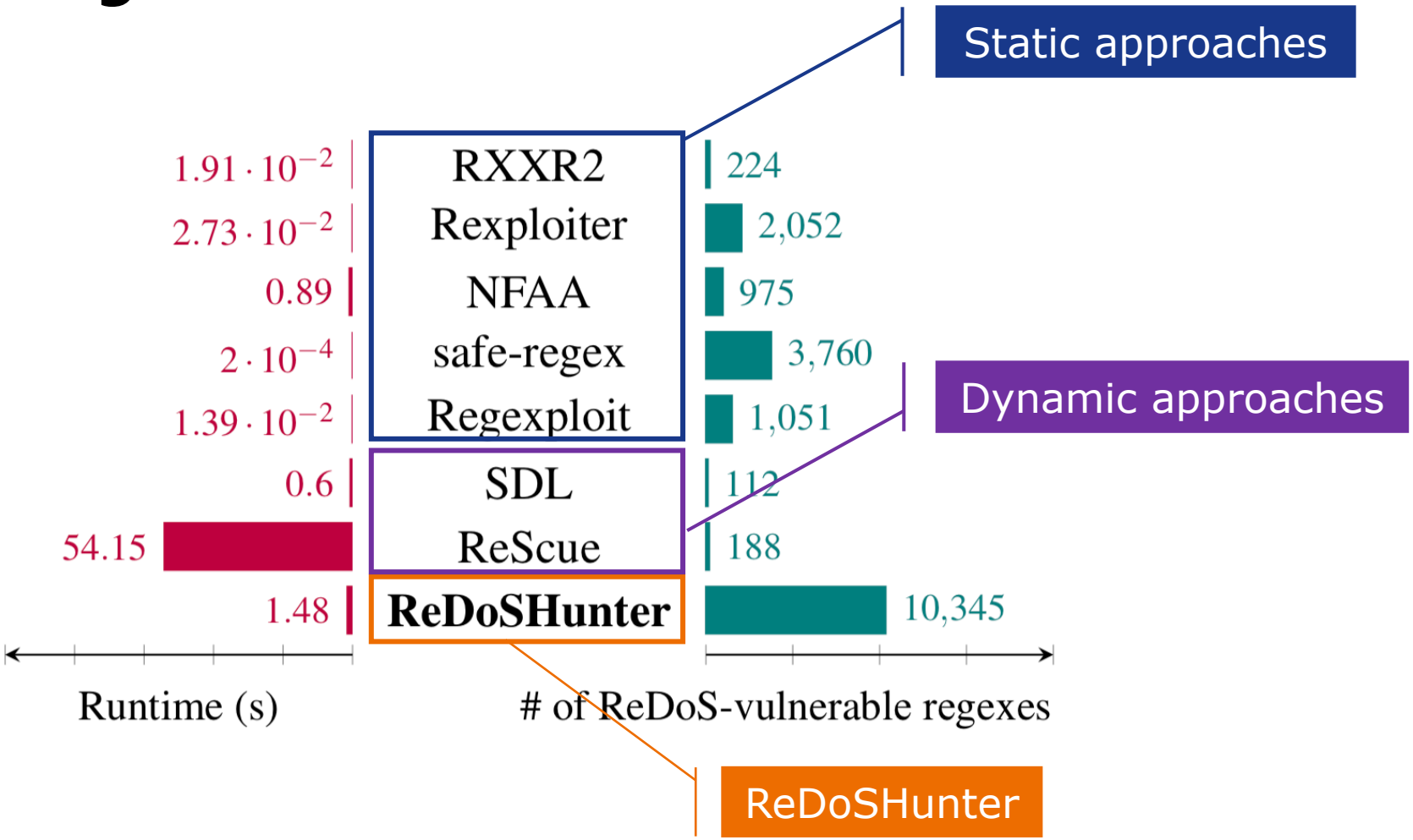
Dynamic approaches

ReDoSHunter

Results on Regex Benchmarks



Results on Regex Benchmarks



Results on Known Vulnerabilities

No.	Project	CVE ID	RXR	RER	NAA	SAX	RET	SDL	RSE	RHR
#1	jquery-validation	CVE-2021-21252	✗	✗	✗	✓	✗	✗	✓	✓
#2	CairoSVG	CVE-2021-21236	✗	✓	✓	✗	✓	✗	✗	✓
#3	date-and-time	CVE-2020-26289	✗	✗	✗	✓	✗	✗	✓	✓
#4	fast-csv	CVE-2020-26256	✓	✗	✓	✓	✗	✗	✓	✓
#5	Python	CVE-2020-8492	✓	✓	✗	✓	✓	✗	✗	✓
#6	websocket-extensions	CVE-2020-7663	✓	✗	✗	✗	✗	✗	✓	✓
#7	websocket-extensions	CVE-2020-7662	✓	✗	✗	✗	✗	✗	✓	✓
#8	url-regex	CVE-2020-7661	✗	✓	✗	✓	✗	✗	✓	✓
#9	uap-core	CVE-2020-5243	✗	✗	✗	✗	✓	✗	✗	✓
#10	waitress	CVE-2020-5236	✗	✗	✗	✓	✓	✗	✗	✓
#11	Cisco IOS	CVE-2020-3408	✓	✓	✓	✓	✓	✓	✓	✓
#12	lodash	CVE-2019-1010266	✗	✗	✗	✗	✗	✗	✗	✓
#13	remarkable	CVE-2019-12041	✓	✗	✓	✓	✓	✓	✓	✓
#14	owasp-modsecurity-crs	CVE-2019-11391	✗	✗	✗	✓	✓	✗	✗	✓
#15	owasp-modsecurity-crs	CVE-2019-11390	✗	✗	✗	✓	✓	✗	✗	✓
#16	owasp-modsecurity-crs	CVE-2019-11389	✗	✗	✗	✓	✓	✗	✗	✓
#17	owasp-modsecurity-crs	CVE-2019-11388	✗	✗	✗	✓	✗	✗	✗	✓
#18	owasp-modsecurity-crs	CVE-2019-11387	✗	✗	✗	✗	✗	✗	✗	✓
#19	highcharts	CVE-2018-20801	✗	✗	✗	✗	✗	✗	✗	✓
#20	uap-core	CVE-2018-20164	✗	✓	✗	✓	✓	✗	✓	✓
#21	js-bson	CVE-2018-13863	✗	✗	✗	✓	✗	✗	✗	✓
#22	nodejs	CVE-2018-7158	✗	✗	✗	✗	✗	✗	✗	✓
#23	protobuf.js	CVE-2018-3738	✓	✓	✗	✓	✓	✓	✗	✓
#24	node-sshpk	CVE-2018-3737	✗	✓	✗	✓	✗	✗	✗	✓
#25	Python	CVE-2018-1061	✗	✓	✓	✗	✗	✗	✗	✓
#26	Python	CVE-2018-1060	✗	✗	✓	✗	✗	✗	✗	✓
#27	brace-expansion	CVE-2017-18077	✗	✗	✗	✓	✗	✗	✗	✓
			✗	✗	✗	✗	✗	✗	✗	✓
			✗	✗	✗	✗	✗	✗	✗	✓
#30	tough-cookie	CVE-2017-15010	✗	✓	✗	✗	✗	✗	✗	✓
#31	jshamcrest	CVE-2016-10521	✓	✗	✓	✓	✗	✗	✓	✓
			✗	✗	✓	✓	✗	✓	✓	✓
			✗	✗	✓	✓	✗	✗	✗	✓
#34	ansi2html	CVE-2015-9239	✓	✗	✓	✓	✓	✓	✓	✓
#35	marked	CVE-2015-8854	✗	✗	✗	✗	✗	✗	✗	✓
		Total	10	9	10	21	12	5	12	**35**
		%	28.57	25.71	28.57	60.00	34.29	14.29	34.29	**100**

Dynamic approaches

Static approaches

ReDoSHunter

Results on Real-world ReDoS-vulnerabilities

No.	Project	Status	RXR	RER	NAA	SAX	RET	SDL	RSE	RHR
#1	ua-parser-js	CVE-2020-7733	✗	✗	✗	✗	✗	✗	✗	✓
#2	trim	CVE-2020-7753	✗	✗	✗	✗	✗	✗	✗	✓
#3	npm-user-validate	CVE-2020-7754	✗	✗	✗	✗	✗	✗	✗	✓
#4	dat.gui	CVE-2020-7755	✗	✓	✗	✗	✓	✗	✗	✓
#5	codemirror-js	CVE-2020-7760	✓	✗	✓	✓	✗	✗	✗	✓
#6	@absolunet/kafe	CVE-2020-7761	✓	✓	✓	✓	✓	✓	✓	✓
#7	express-validators	CVE-2020-7767	✓	✗	✗	✓	✓	✗	✗	✓
#8	djvalidator	CVE-2020-7779	✓	✓	✓	✓	✓	✓	✓	✓
#9	ua-parser-js	CVE-2020-7793	✗	✗	✗	✗	✗	✗	✗	✓
#10	glob-parent	CVE-2020-28469	✗	✗	✗	✗	✓	✗	✗	✓
#11	jinja2	CVE-2020-28493	✗	✓	✓	✗	✗	✗	✗	✓
#12	three	CVE-2020-28496	✗	✓	✓	✗	✗	✗	✗	✓
#13	lodash	CVE-2020-28500	✗	✗	✗	✗	✗	✗	✗	✓
#14	py	CVE-2020-29651	✗	✗	✗	✗	✗	✗	✗	✓
#15	uap-core	CVE-2021-21317	✗	✗	✗	✗	✗	✗	✗	✓
#16	CKEditor 5	CVE-2021-21391	✗	✓	✓	✗	✗	✗	✗	✓
#17	prism	CVE-2021-23341	✗	✗	✗	✗	✗	✗	✗	✓
#18	path-parse	CVE-2021-23343	✗	✗	✗	✗	✗	✗	✗	✓
#19	html-parse-stringify	CVE-2021-23346	✓	✓	✗	✓	✓	✓	✗	✓
#20	jspdf	CVE-2021-23353	✓	✗	✓	✓	✗	✗	✗	✓
#21	printf	CVE-2021-23354	✗	✗	✗	✓	✗	✗	✗	✓
			✗	✗	✗	✓	✗	✗	✗	✓
			✗	✗	✓	✓	✗	✗	✗	✓
#24	postcss	CVE-2021-23368	✗	✗	✓	✗	✗	✗	✓	✓
#25			✗	✗	✗	✗	✗	✗	✗	✓
#26			✗	✗	✓	✓	✓	✗	✓	✓
#27			✗	✗	✗	✗	✗	✗	✓	✓
#28	validator	Fixed	✗	✗	✗	✗	✗	✗	✓	✓
		Total	7	8	10	9	7	3	6	**28**
		%	25.00	28.57	35.71	32.14	25.00	10.71	21.43	**100**

Dynamic approaches

Static approaches

ReDoSHunter

Summary

- We proposed ReDoSHunter, reaching 100% precision and 100% recall over three Large-scale datasets.
- We proposed five ReDoS patterns that are identified from our massive investigation and analysis.
- We exposed 28 new ReDoS-vulnerabilities in popular open-source projects with **26 assigned CVEs** and 2 fixed by the maintainers.

THANKS

Q&A