35th USENIX Security Symposium

Regular Expression Denial of Service Induced by Backreferences

Yichen Liu Berk Çakar Aman Agrawal Minseok Seo
James C. Davis Dongyoon Lee

Stony Brook University

COSY Compiler · OS · Security

PURDUE UNIVERSITY

DUALITY LAB

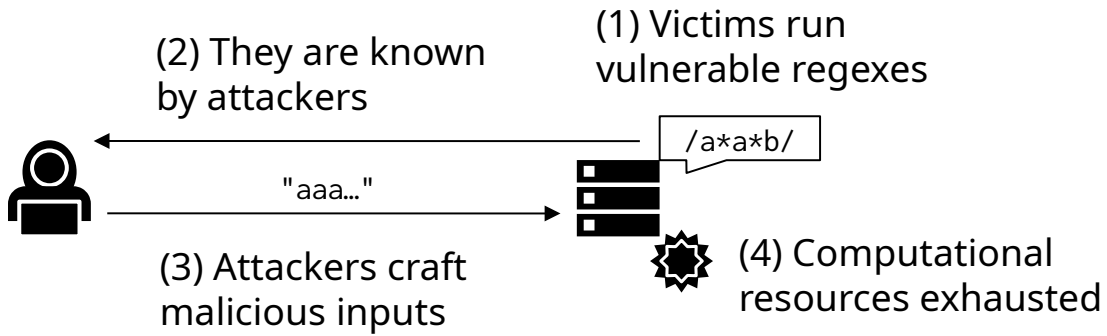
NSF Funded Research

August, 2026
Long Presentation

Background

Regular Expressions Denial of Service (REDoS)

- **Regular Expression (regex)**: languages for string matching.
- **Regex Backtracking Runtime Complexity**:
  - Most linear ($O(n)$), but some **superlinear**.
- **REDoS**: complexity DDoS caused by such runtime.
- **REDoS Threat Model:**

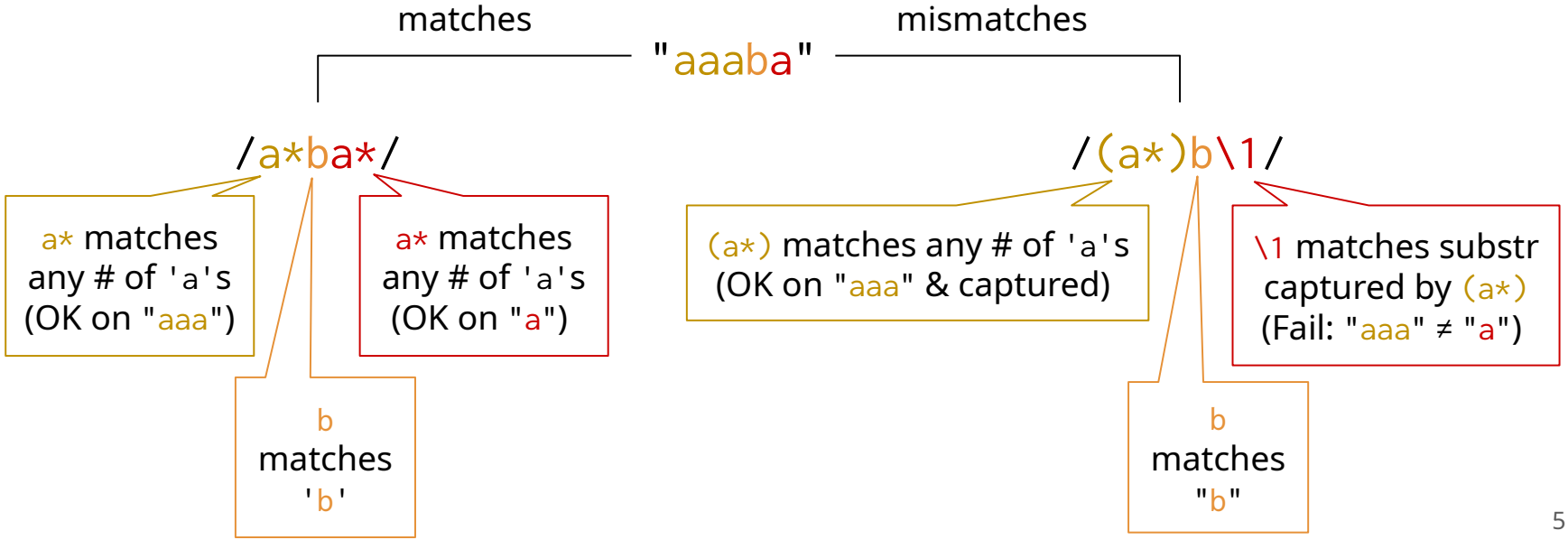


Regular Expressions Types

- **Kleene** (regular/classical) **regex**: can be converted to **NFAs**
  - Empty string ε
  - Character σ
  - Concatenation rr
  - Alternation $r|r$
  - Repetition $r*$
- Modern regex engines equip **irregular** features (no equivalent NFAs).
  - Capture group (r), backreference \i
  - Look around assertions, atomic groups, …

Regular Expressions with Backreferences (REwBs)

- **REwBs** contain:
  - **Capture groups** (*r*) : store the substrings matched by sub-regex *r*.
  - **Backreferences** \i : match exactly the substring captured by group *i*.
- **E.g.**: /a*ba*/ matches any "$a^x ba^y$". /(a*)b\1/ only matches "$a^x ba^x$".



REDoS Static Detection and Gaps

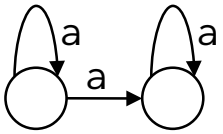
Kleene Regex:

Detecting double overlap loops finds all superlinear Kleene regexes.

≡

NFA:

Superlinear Runtime ⇒ **Superlinear Sink Ambiguity** ⇔ **Double Overlap Loops**



Kleene regex equivalent to NFA

NFA runtime is bounded by sink ambiguity

NFA superliner sink ambiguity iff double overlap loops

Overlap: accept same chars

Regex with Backreference (REwB):

Used to be no static detector for REwB.

≠

MFA:

Superlinear Runtime ? **Sink Ambiguity** ? **Double Overlap Loops**

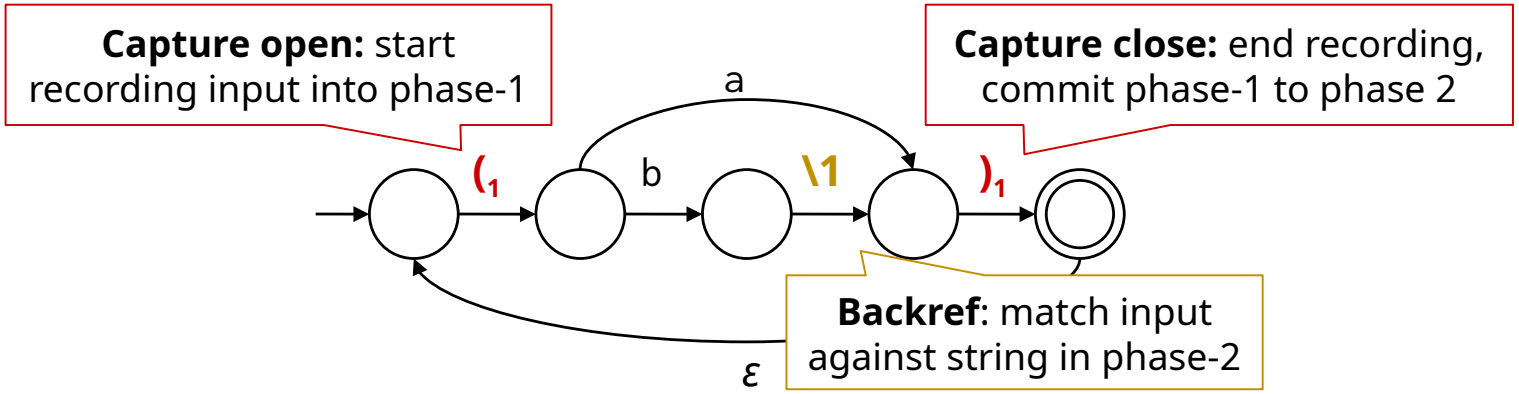
Existing Memory Automaton (MFA) cannot model some REwBs

Unknown relation between REwB runtime and sink ambiguity

Unknown relation between REwB runtime ambiguity, and structures

Our Contributions

Two-phase Memory Automaton (2PMFA)

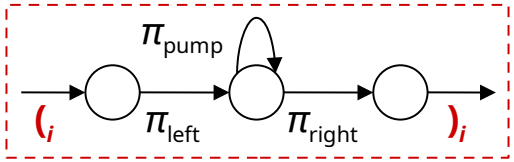


	Phase-1 Memory	Phase-2 Memory
1	...	...

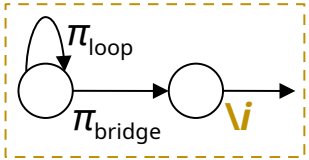
Compared to NFA: support capture and backref
Compared to MFA: one more phase of memory to support **self-reference**

Theorems for Detection

- **Claim**: 2PMFA runtime may not be bounded by sink ambiguity.
- **Theorem 1**: If Claim is true, there must be a backreference that:
  (1) Costs non-*O*(1) in each evaluation, AND,
  (2) Evaluates non-*O*(1) number of times.
- **Lemma 1**: If (1), there is a loop or backref in the referred group.
- **Lemma 2**: If (2), there is a loop before the backref.



Loop in
capture group

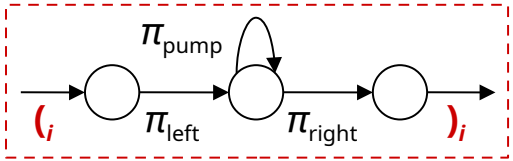


Loop before
backreference

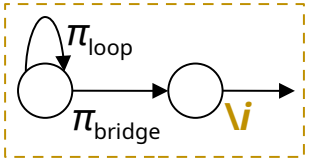
- Note: backref needs string comparison, takes *O*(*n*), not *O*(1).
- Intuition: *O*(*n*) cost each × non-*O*(1) evals → non-*O*(*n*) runtime.

Theorems for Detection

- **Theorem 2**: If 2PMFA runtime is not bounded by sink ambiguity, then there are:
  (1) A loop or backref in the referred group, AND
  (2) A loop before the backref.
- Cause vulnerabilities that previous detectors cannot find.



Loop in
capture group



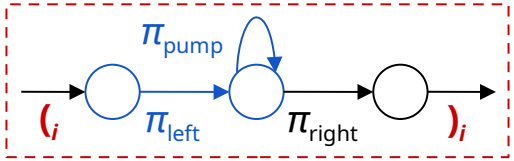
Loop before
backreference

Before deriving 3 vulnerability patterns from them:

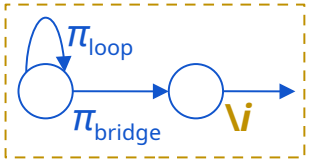
- Assume **no backrefs in groups**.
- Assume **linear sink ambiguity**

Vulnerability Pattern 1

If 2PMFA with **linear sink ambiguity** exhibit **superlinear runtime**:



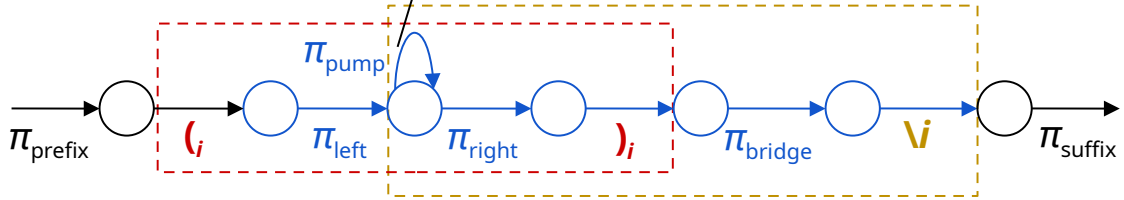
Loop in capture group



Loop before backreference

(1) If two loops are the same one.

Pattern 1



π_{left}, π_{pump}, $\pi_{right}\pi_{bridge}$ overlap

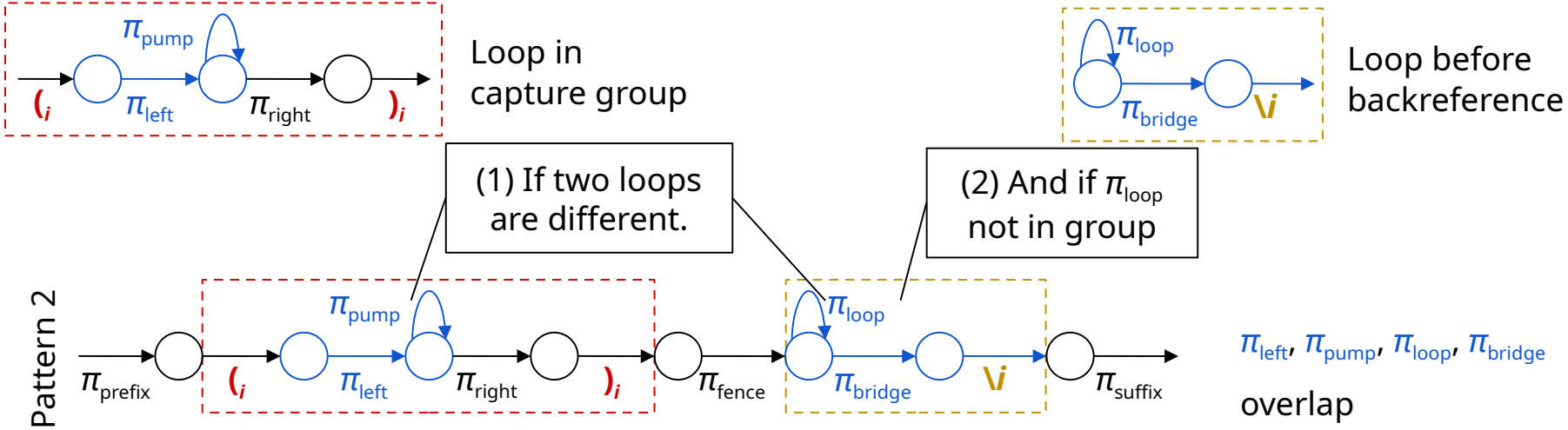
Example: run `/(a*)\1b/` on "a^{2n}":

Backtrack #	(a*) match	\1 match
0	"a^0"	"a^0"
1	"a^1"	"a^1"
…	…	…
n	"a^n"	"a^n"
…	…	…

Total cost by \1
$= 0 + \ldots + n = n\,(n+1) / 2$
$\in O(n^2)$

Vulnerability Pattern 2

If 2PMFA with **linear sink ambiguity** exhibit **superlinear runtime**:



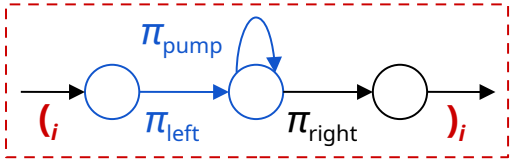
Example: run `/(a*)ba*\1b/` on "$a^n ba^{2n}$":

Backtrack #	(a*) match	a* match	\1 match
0	"a^n"	"a^0"	"a^n"
1	"a^n"	"a^1"	"a^n"
…	…	…	…
n	"a^n"	"a^n"	"a^n"
…	…	…	…

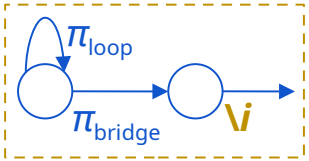
Total cost by \1
$= n\,(n+1)$
$\in O(n^2)$

Vulnerability Pattern 3

If 2PMFA with **linear sink ambiguity** exhibit **superlinear runtime**:



Loop in capture group

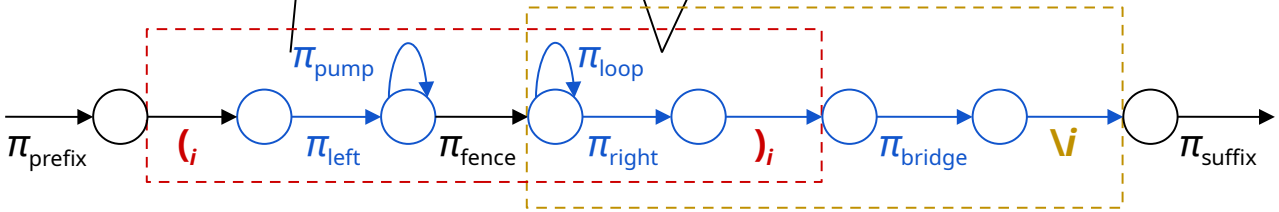


Loop before backreference

(1) If two loops are different.

(2) And if π_{loop} in group

Pattern 3



π_{left}, π_{pump}, π_{loop}, $\pi_{right}\pi_{bridge}$ overlap

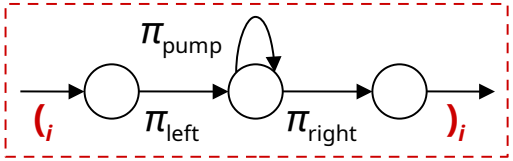
Example: run `/(a*ba*)\1b/` on "$a^n ba^{2n}$":

Backtrack #	(a* match	a*) match	\1 partially match
0	"a^n"	"a^0"	"a^n"
1	"a^n"	"a^1"	"a^n"
…	…	…	…
n	"a^n"	"a^n"	"a^n"
…	…	…	…

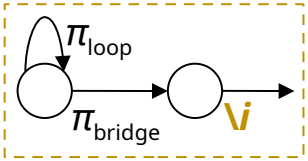
Total cost by \1
$= n\,(n+1)$
$\in O(n^2)$

Three Vulnerability Patterns

If 2PMFA with **linear sink ambiguity** exhibit **superlinear runtime**:

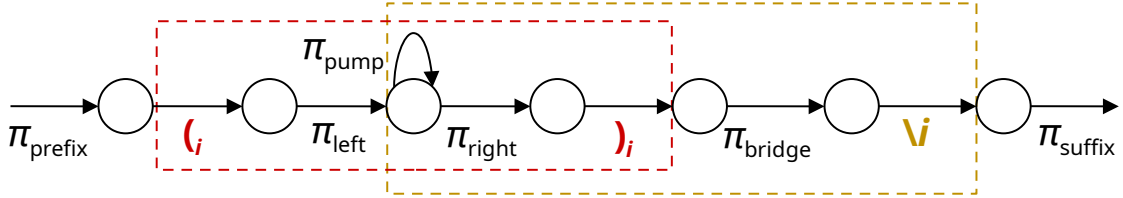


Loop in capture group



Loop before backreference

Pattern 1

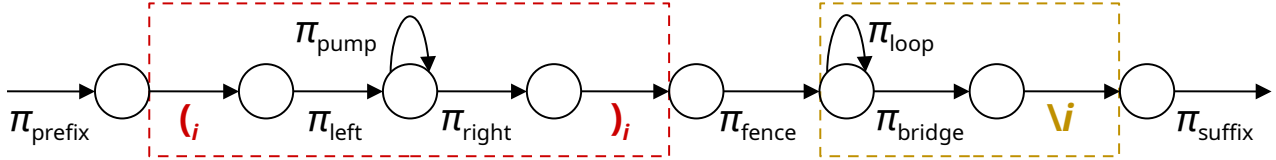


$\pi_{pump} = \pi_{loop}$

π_{left}, π_{pump}, $\pi_{right}\pi_{bridge}$ overlap

Example: `/(a*)\1b/`

Pattern 2

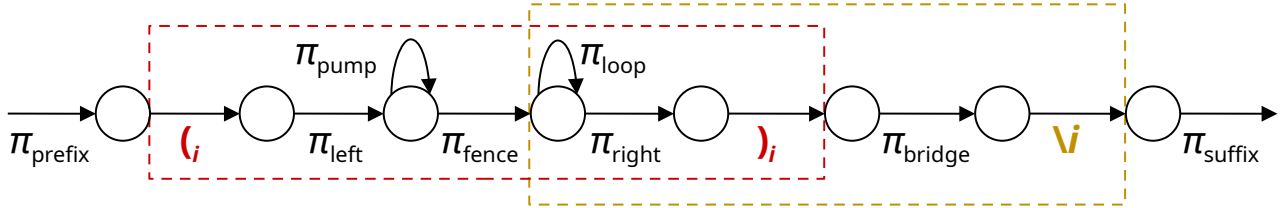


$\pi_{pump} \neq \pi_{loop}$ | π_{loop} not in group

π_{left}, π_{pump}, π_{loop}, π_{bridge} overlap

Example: `/(a*)ba*\1b/`

Pattern 3

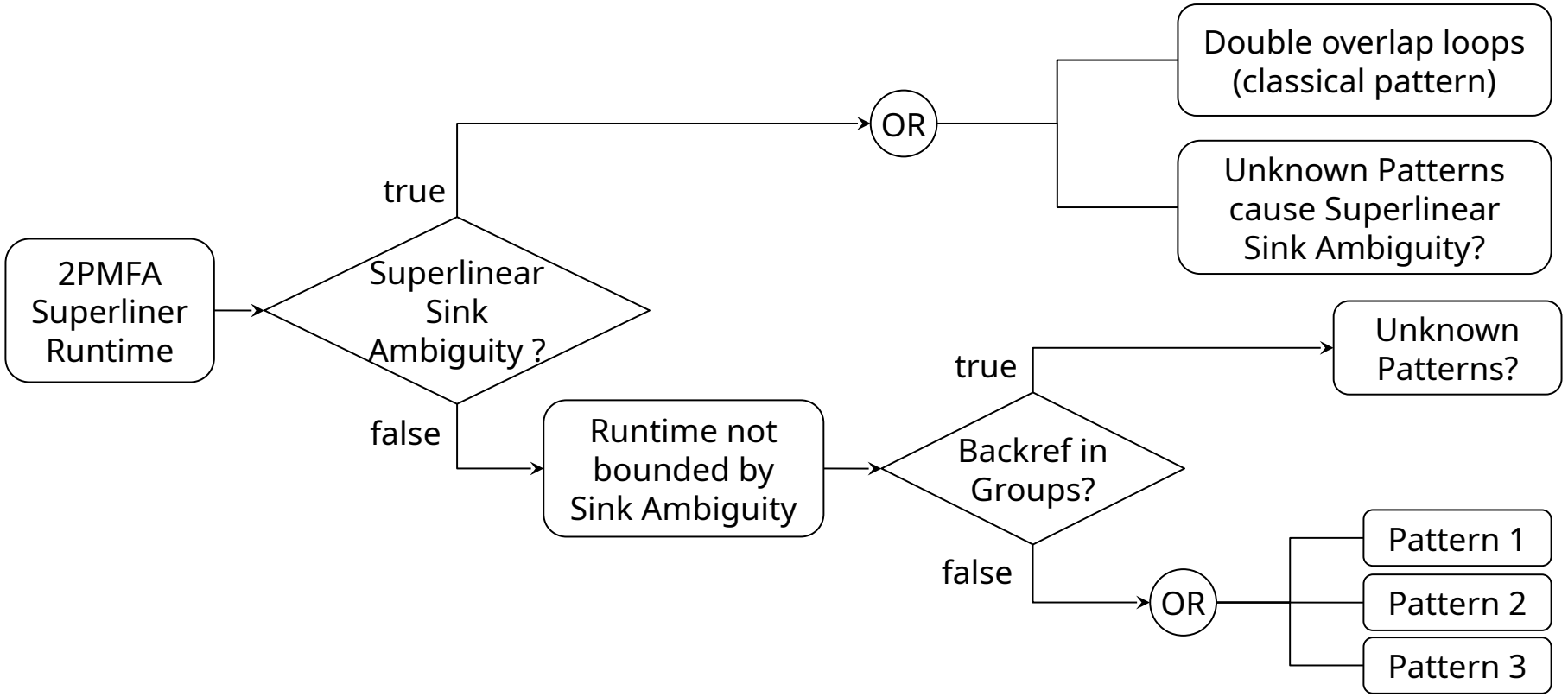


$\pi_{pump} \neq \pi_{loop}$ | π_{loop} in group

π_{left}, π_{pump}, π_{loop}, $\pi_{right}\pi_{bridge}$ overlap

Example: `/(a*ba*)\1b/`

Necessity Logics



Evaluation

Detection

- We implemented a detector “**REwBGuard**”.
- Found **48** vulnerable regexes out of 11,266 from **Snort** 2.9 rule set.
- Their runtime are measured on PCRE, Python, Java, and Onigmo, which most exhibit $\Omega(n^2)$ or higher complexity.

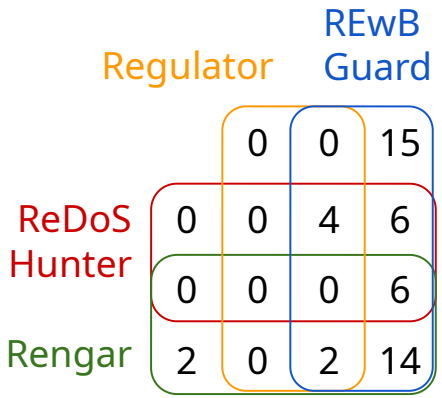
	Only	**Also 2 Overlap Loops**	**Total**
Pattern 1	1	8	9
Pattern 2	11	28	47
Pattern 3	0	0	0
Pattern 1-3	12	36	48

REwBGuard Repository:
https://github.com/SBULeeLab/rewbguard

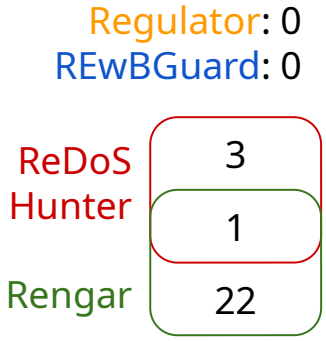
Comparison

- REwBGuard found exclusive vulnerabilities in **15** regexes comparing to state-of-the-art detectors.
- Previous ones occasionally detect our patterns after replacing backrefs by capture groups, but introduces false positives.

Detection Comparison by Regexes



True Positives

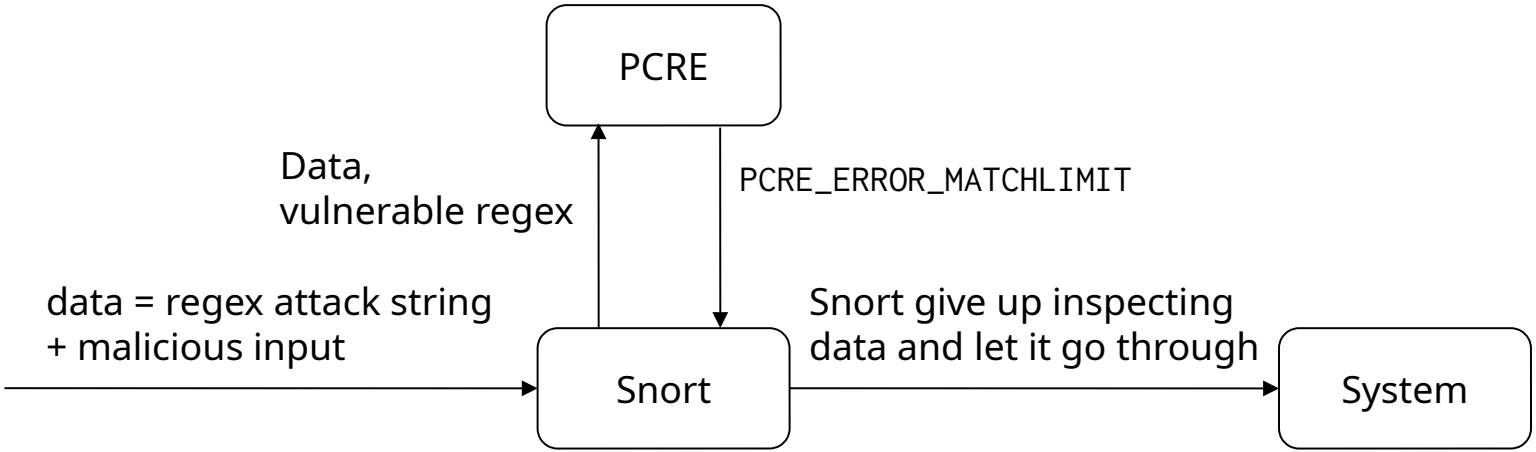


False Positives

REwBGuard is our detector

Exploit Snort

- We developed 4 attacks targeting Snort 2.9 using found vulnerabilities.
- Attack type 1: slowing down (0.6-1.2 sec).
- Attack type 2: suppressing alert or dropping.
- Exploits are reported to and reviewed by Snort developers / Cisco PSIRT.



Conclusion

- NFA ambiguity based DoS detectors are not enough for REwB.
- New model 2PMFA for REwB.
- 2PMFA runtime may be not bounded by sink ambiguity, and there are necessary condition where this happens.
- 3 vulnerability patterns.
- Evaluation:
  - Detected 48 super-linear regexes.
  - Developed 4 attacks targeting Snort.

Thank You



Contact: Yichen Liu, Dongyoon Lee <{liu38,dongyoon}@cs.stonybrook.edu>

Paper & Presentation:
https://www.usenix.org/conference/usenixsecurity26/presentation/liu-yichen