

RAID 2026

Vulnerability Evolution and the Promise of Automated Gatekeeping in Open-Source Software

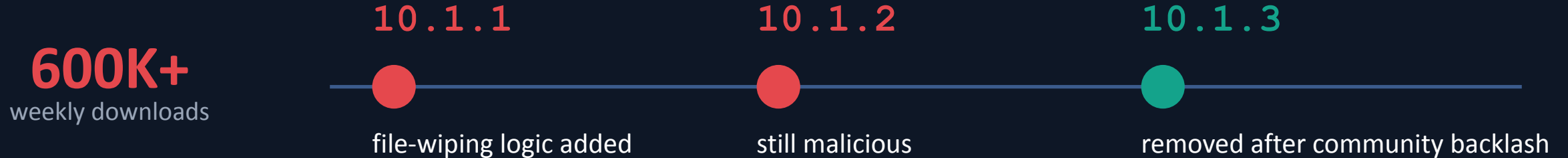
Presenter: Ali Akhavani



Northeastern University
Systems Security Lab



A popular npm package started overwriting its users' files



Shipped by the package's own maintainer: no typosquat, no stolen credentials



Looked up each user's IP location and targeted Russia and Belarus as a protest



Overwrote files with a heart emoji; classified as CWE-506, embedded malicious code



```
// node-ipc 10.1.1 (simplified)
const country = geoip(myPublicIP())
if (["RU", "BY"].includes(country)) {
  for (const f of walk("/"))
    overwrite(f, "♥")
}
```

node-ipc

A backdoor two years in the making

2021



“Jia Tan” starts contributing helpful patches to xz

2022



Sock-puppet accounts pressure the lone, unpaid maintainer to accept help

2023



Jia Tan becomes a trusted co-maintainer

2024



Backdoored 5.6.0 and 5.6.1 ship: pre-auth RCE through sshd

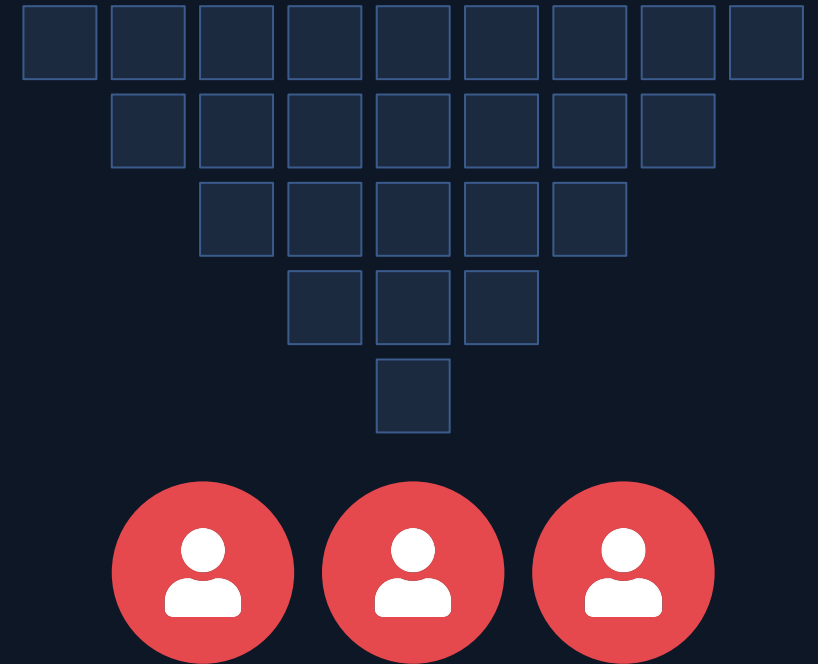


March 29, 2024: an engineer noticed SSH logins ~500 ms slow. Caught by chance, before most stable releases.



No matter how trusted an open-source package is, it does not guarantee that its future releases are trusted.

Are these outliers or symptoms?



Three questions



RQ1

Remediation gap

Can maintainers keep up with the growth of reported vulnerabilities?



RQ2

Risk patterns

Where does risk concentrate, and how do malicious packages get in?



RQ3

LLM gatekeeping

Can LLMs flag vulnerable code before it is published?

Nine years, ten languages, one dataset

30,572

vulnerability reports

16,802

affected packages

2017–2025

through late 2025

8 package managers + C/C++ libraries

npm

PyPI

Maven

Packagist

Go

Crates

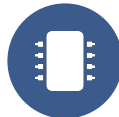
NuGet

RubyGems

C/C++



5× larger than prior
package-vulnerability studies



First to include C/C++ package
vulnerabilities



Open dataset:
github.com/fiu-seclab/oss-security

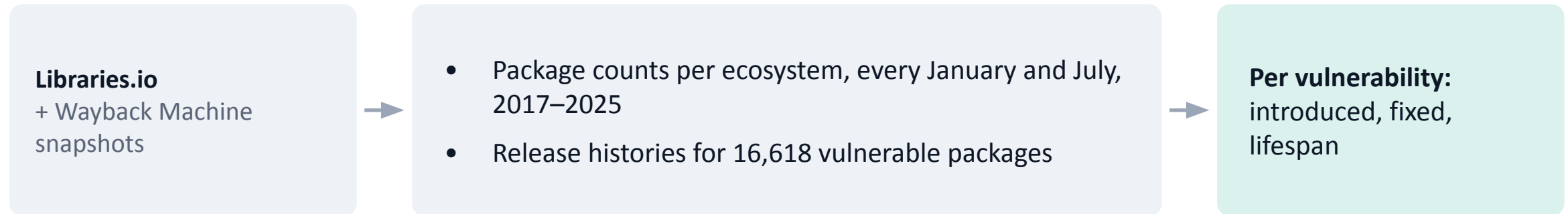
Extends our earlier preprint, Open Source, Open Threats? [3]

Converting raw advisories to a unified view

VULNERABILITY REPORTS



ECOSYSTEM HISTORY



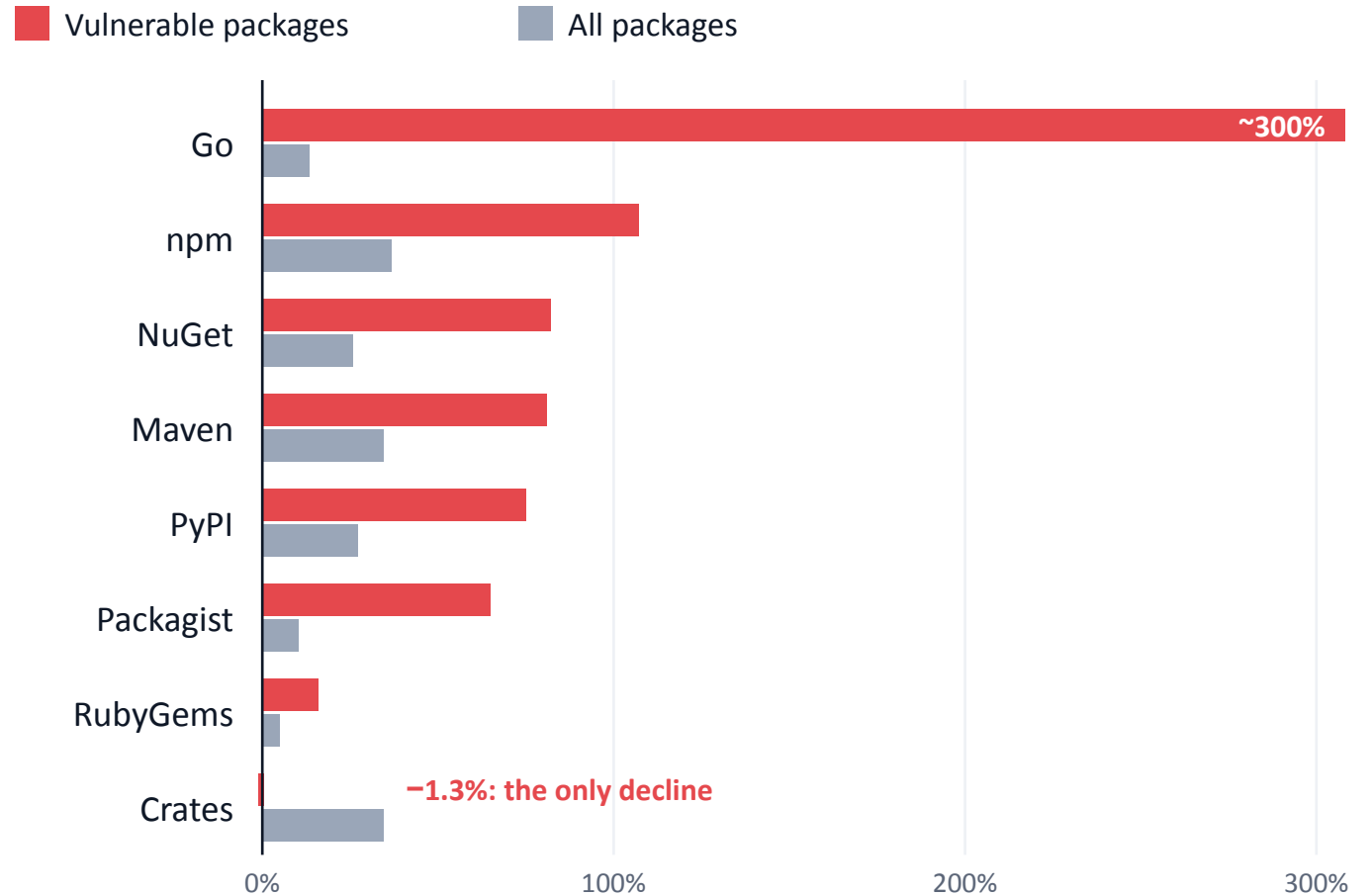
Vulnerabilities are outgrowing the ecosystems

+91.5%

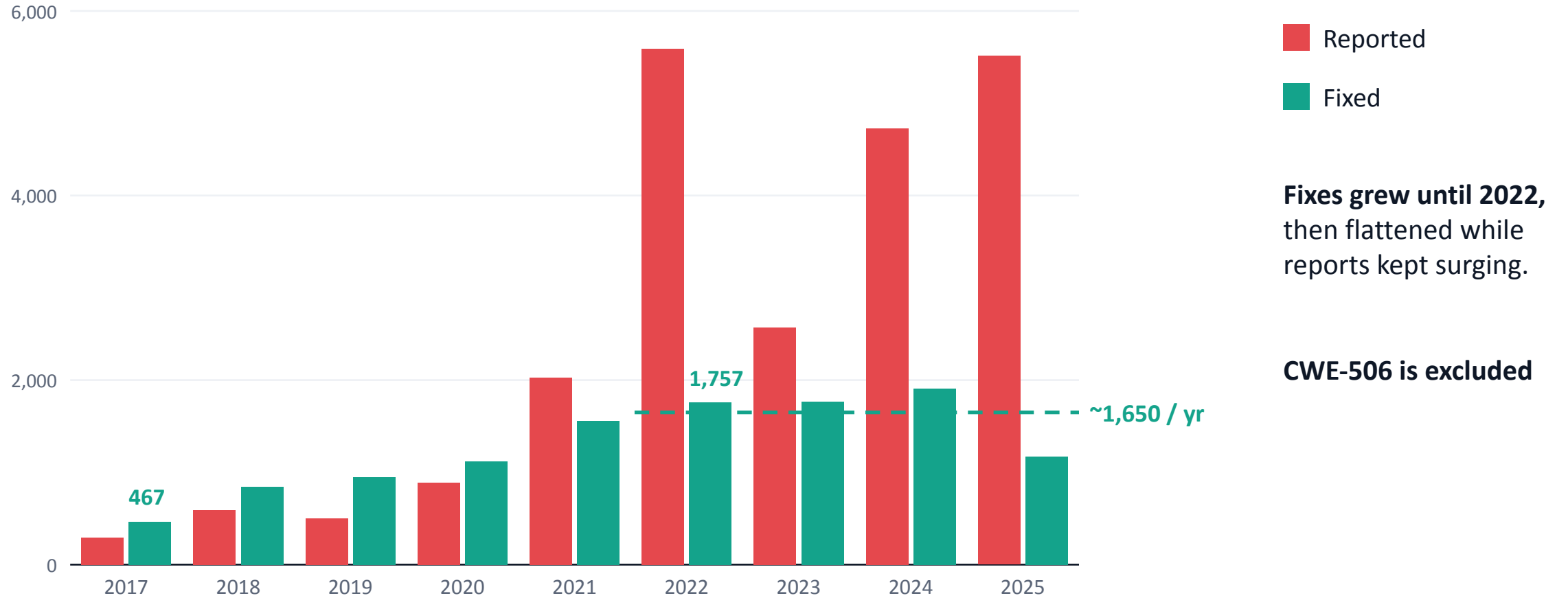
average annual growth of packages with reported vulnerabilities

+26%

average annual growth of all packages

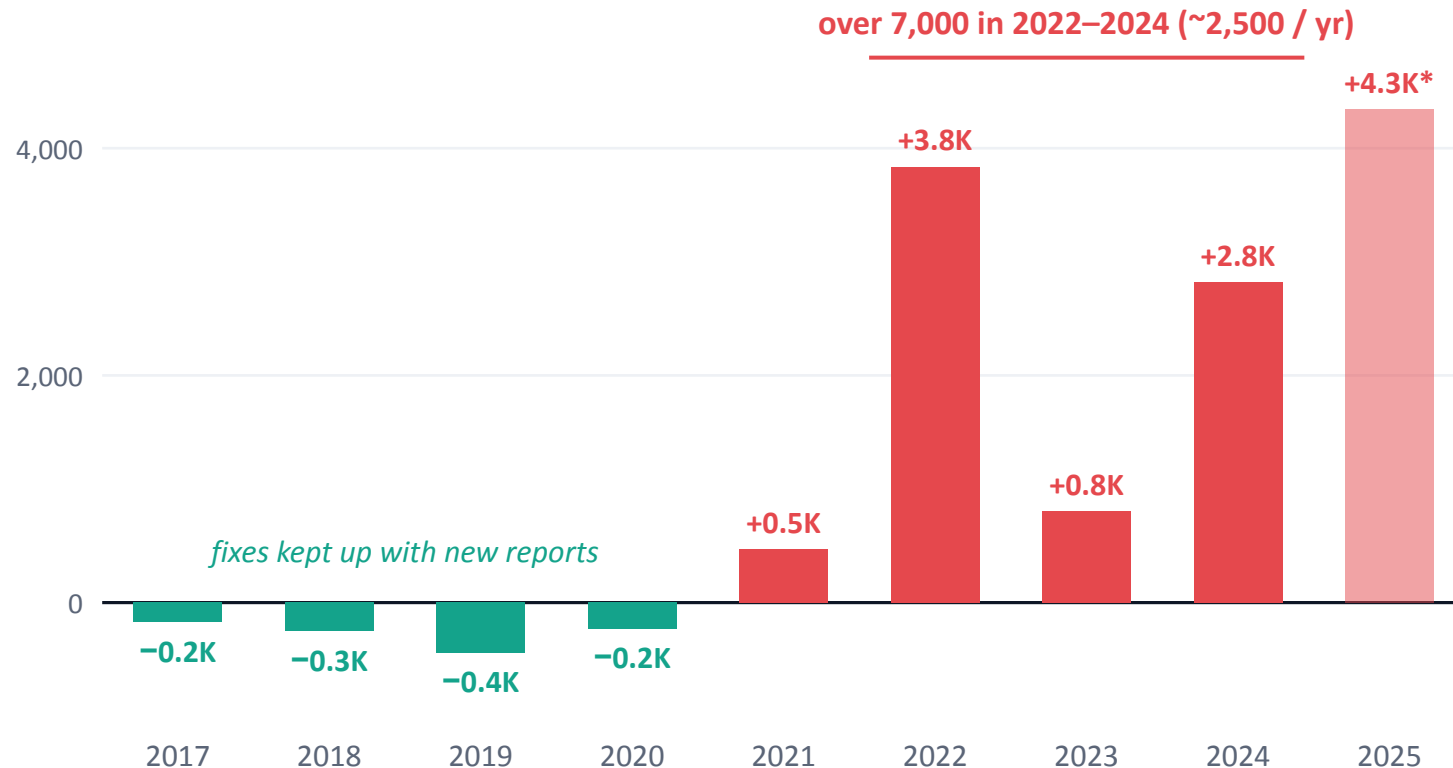


Fixes hit a ceiling



Since 2021, reports outpace fixes every year

Yearly gap = newly reported – fixed (CWE-506 excluded)



Counts read from Fig. 8. * 2025 runs through late 2025.

How it's measured

Reported: advisories published that year, excluding malicious packages (CWE-506)

Fixed: vulnerabilities whose first patched version was released that year

Gap: reported – fixed; above zero means the backlog grew

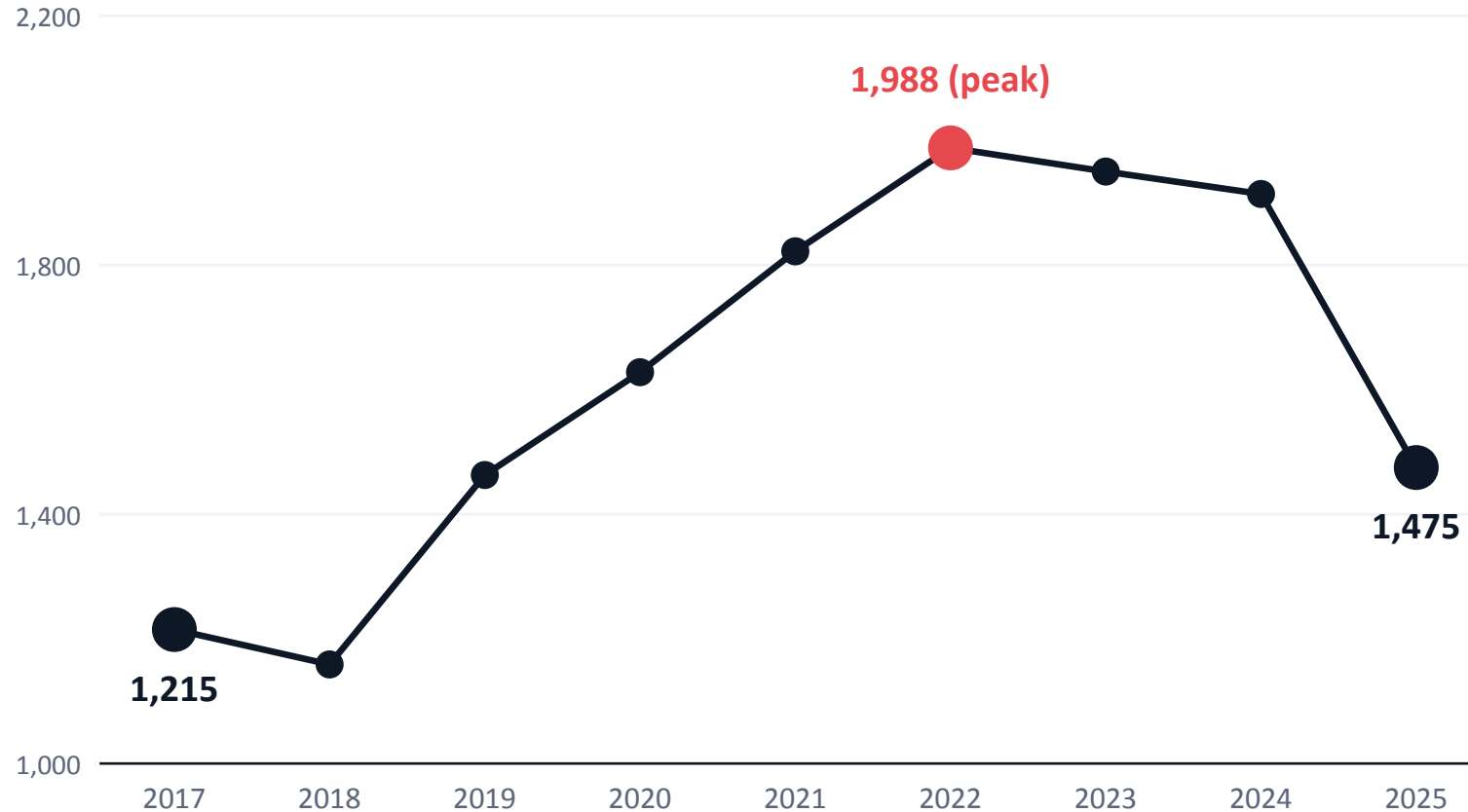
0.44 fixes per reported vulnerability
2020–2025 combined

7,000+ net gap in 2022–2024

A net shortfall, not a list of open advisories: a fix can close a vulnerability reported in an earlier year.

Vulnerabilities live for years

Average lifespan (days), by fix year



~4 years

average lifespan of vulnerabilities fixed in 2025, still 21% above 2017

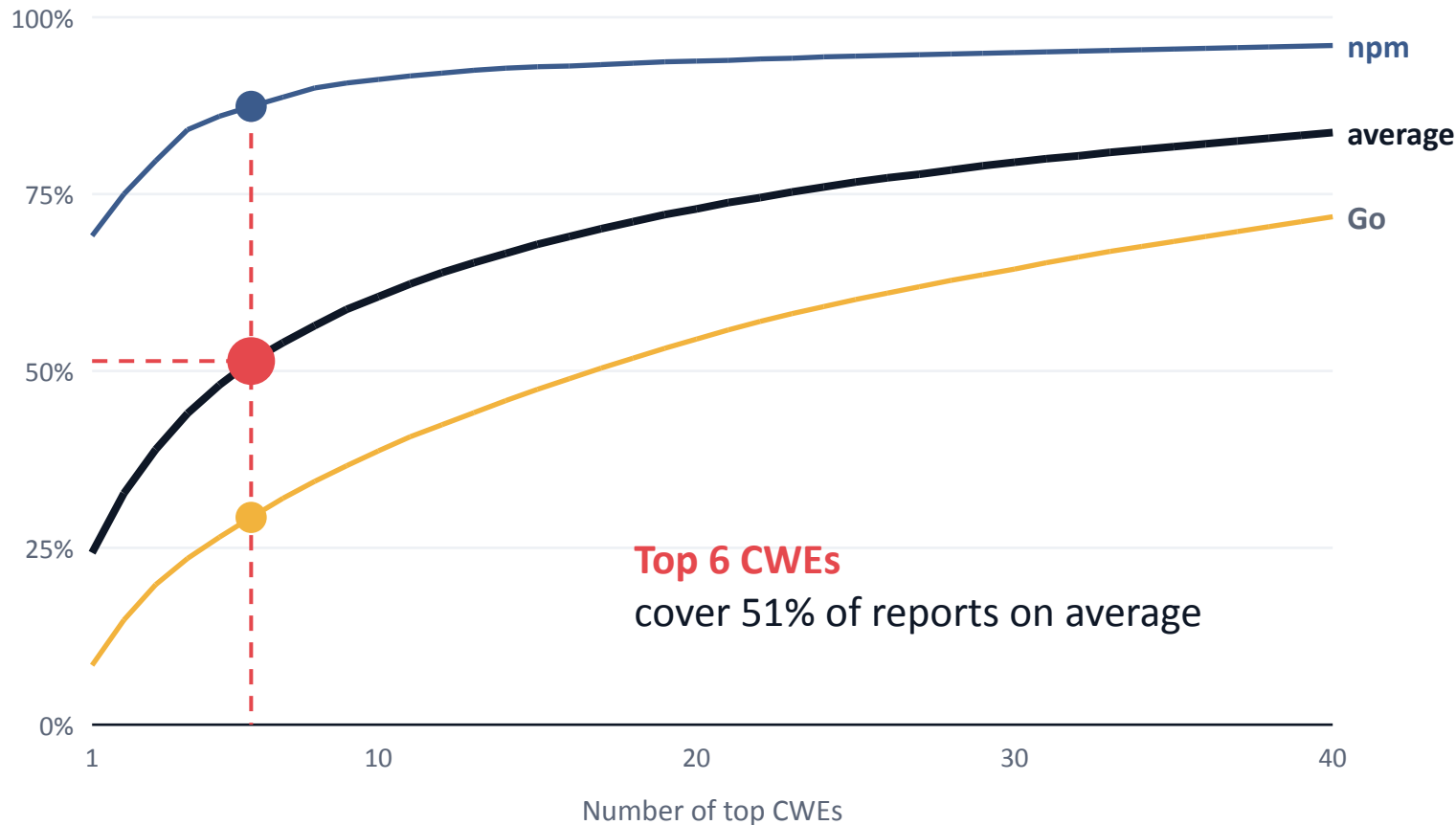


2025 fixes favored newly introduced bugs; older ones were left behind

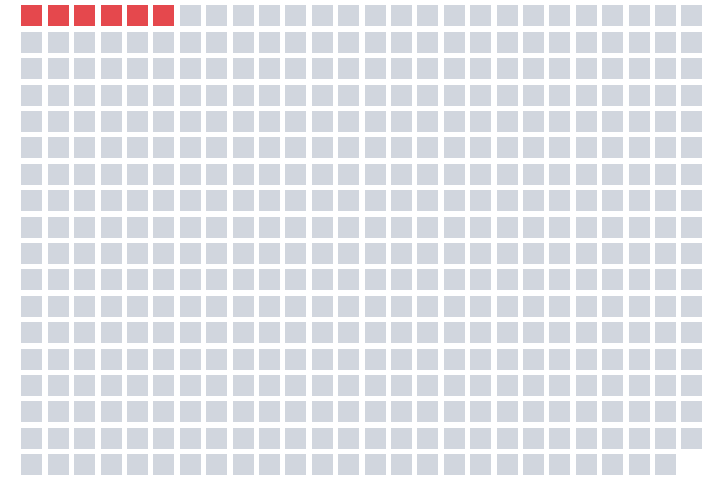
Lifespan: first affected release → first patched release, summed over affected ranges.

467 distinct CWEs but just 6 cover half the reports

Cumulative share of reports covered by the top-N CWEs



467 distinct CWEs in the dataset

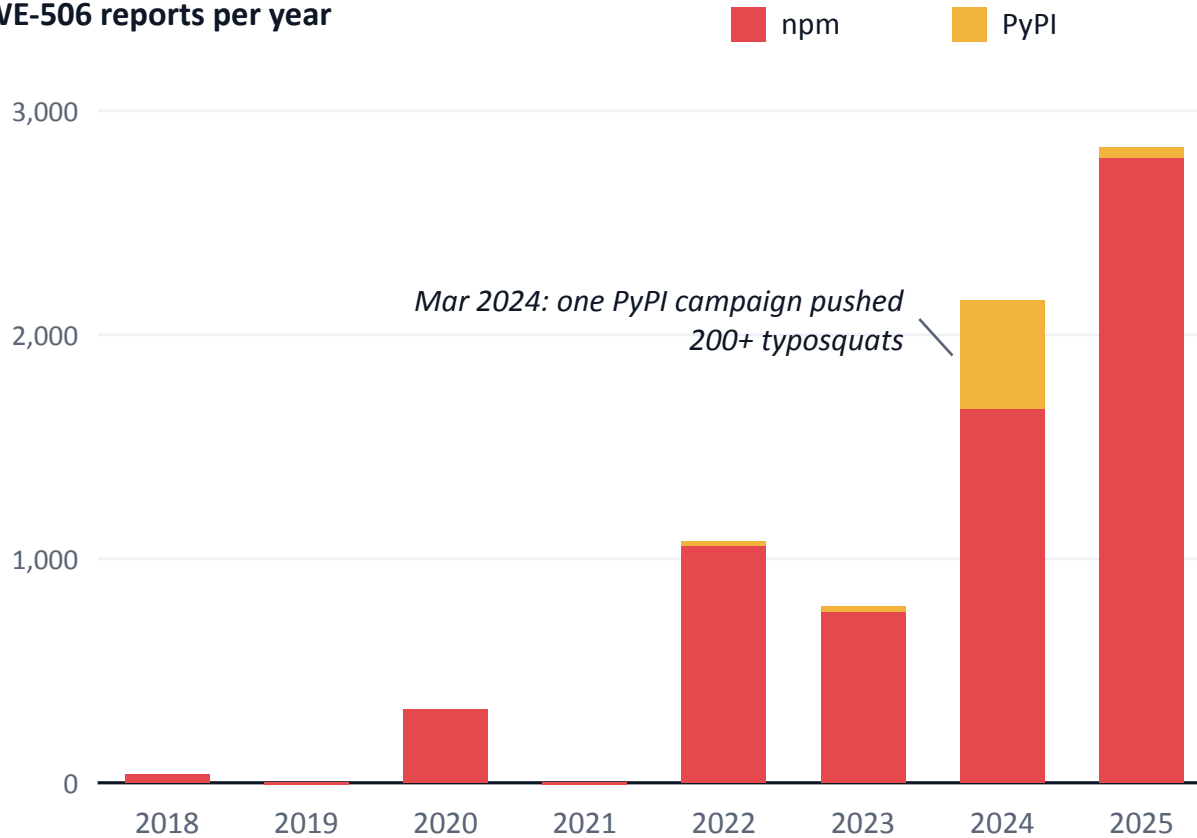


6 of them cover over half of a typical ecosystem's reports

Top 6 cover ~87% in npm, ~29% in Go.

Malicious packages: 38 reports in 2018, 2,952 in 2025

CWE-506 reports per year



Bars: npm and PyPI, read from Fig. 10. Registries other than npm and PyPI add 141 reports in total.

Where they land

Registry	Reports	Share of the registry's own reports
npm	6,679	67.2%
PyPI	578	14.0%
RubyGems	64	8.1%
NuGet	40	5.6%
Go	19	0.8%
Crates	15	1.5%

npm + PyPI: 98% of all 7,398 malicious-package reports

Source: Table 3. Remaining registries: 3 reports.

How malicious packages get in



Typosquatting

`lodahts`

`jquery.js`

`tensofloww`

`pygaqme`

11.4% of malicious names closely mimic popular libraries



Dependency confusion

`@swiggy-private/analytics`

1,156 scoped packages (15.6%) impersonate internal components



Install hooks

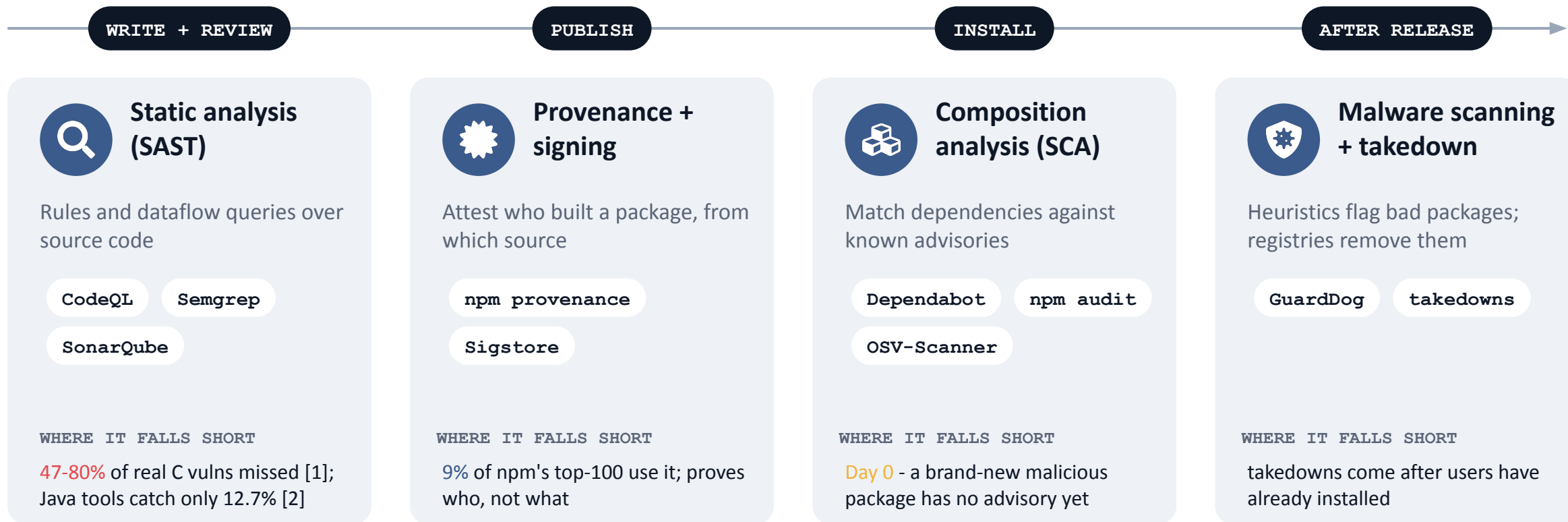
`preinstall`

`postinstall`

Code runs at install time, before the package is ever imported

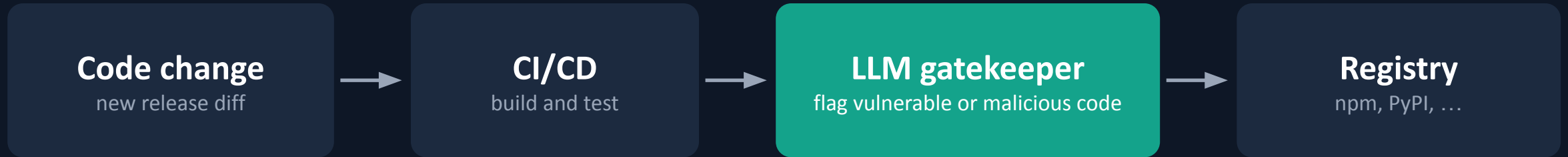
None of these is a coding bug: they abuse trust in names, scopes, and install-time behavior.

Today's defenses, and where they fall short



Missing: a check that reads what a new release actually does, at publish time, in any language

What if every release passed a gatekeeper?



Humans don't scale

Fixes plateaued at ~1,650 / yr while reports surge (RQ1)



Risk is concentrated

~6 CWEs cover half of all reports in a typical ecosystem (RQ2)



Today's tools miss it

SAST misses most real bugs; SCA and provenance can't judge new code

Testing on real version diffs, not snippets



1 Any of the top 5?

One prompt per diff lists the ecosystem's five most reported CWEs

2 Is it CWE-X?

One prompt checks a single CWE and must cite the exact lines

582

version pairs

1,164

labeled samples

1,434

advisories covered

≤5K

changed lines (~32K tokens)

10

LLMs (2 API, 8 open-weight)

What we ask the model

1 Any of the top 5?

per version diff

You are a security code reviewer analyzing **{language}** code, provided with a diff file that shows the changes between the OLD and NEW code.

TASK: Determine if adding the NEW code (lines marked with "+") and removing the OLD code (lines marked with "-") would introduce ANY of these vulnerabilities:

{top-5 CWEs for this ecosystem}

DIFF TO ANALYZE: **{diff}**

Respond with ONLY this JSON:

```
{"vulnerable": true/false, "cwe_found": "CWE-XXX" or null, "evidence": "..."}

```

Used on 582 version pairs across 10 ecosystems

2 Is it CWE-X?

per source file

You are a security code reviewer analyzing **{language}** code. Target file: **{filename}**

ONLY evaluate for **CWE-506 Embedded Malicious Code**. Do NOT report any other types of issues.

- If a CWE cannot be conclusively identified, do NOT include it.
- Every finding MUST include precise line numbers, the exact code snippet, and why it matches the CWE.

Output JSON only:

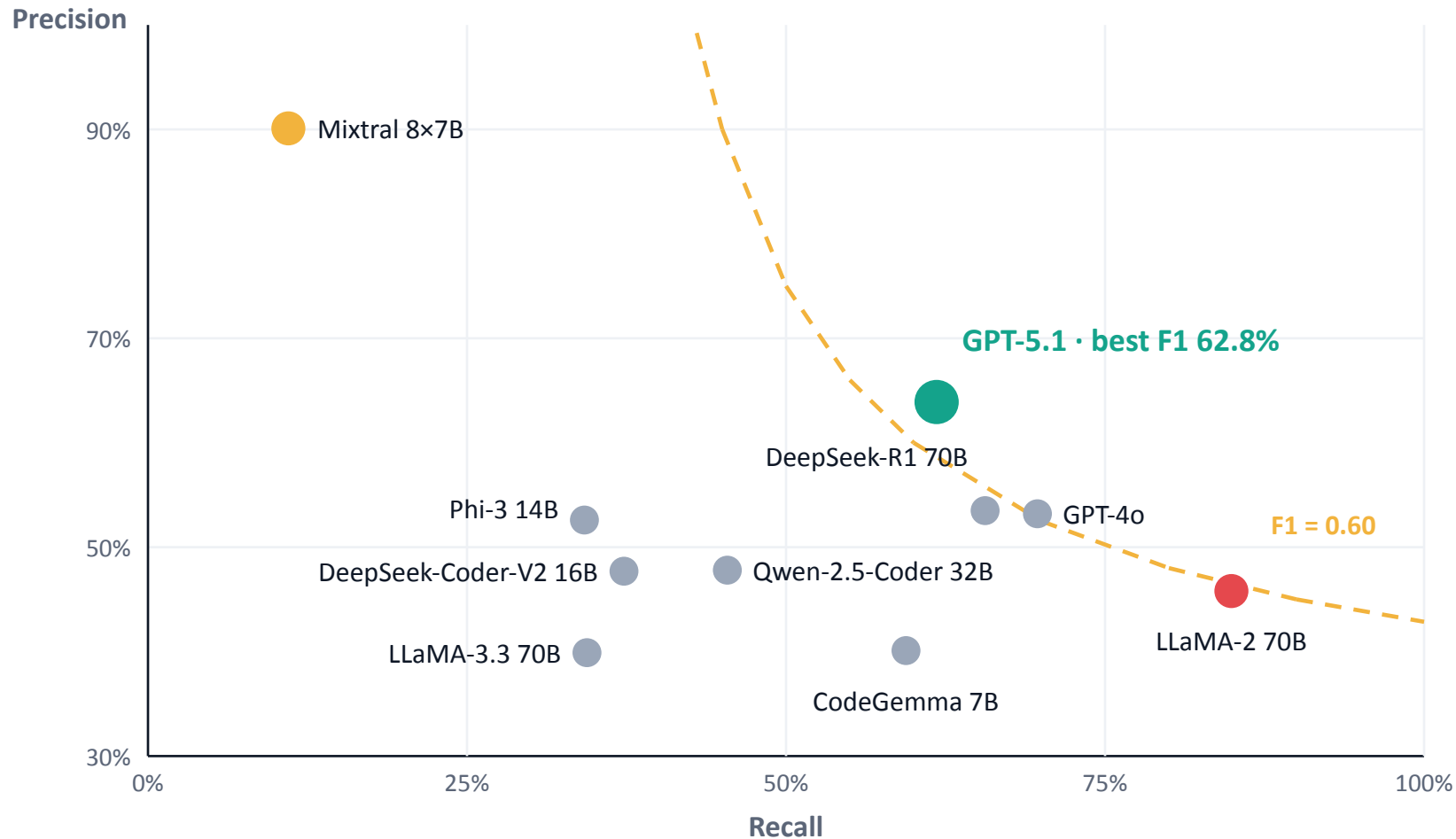
```
{"vulnerabilities_found": [ ... ], "safe": true|false}

```

Used on 600 confirmed malicious packages (Datadog)

Negatives use the same prompt pointed at the removed lines. Prompts condensed from Listings 1.1 and 1.2.

No model wins on both precision and recall



- **GPT-5.1** most balanced: 63.9% precision, 61.8% recall
- **LLaMA-2 70B** catches 84.9%, but 54% of its alerts are false
- **Mixtral 8x7B** 90.1% precision, yet misses 89% of vulnerabilities

Best F1 is on par with prior work (57–66%), here on real diffs across 10 ecosystems.

Models over-predict XSS

CodeGemma 7B · F1 47.9%

	79	22	1321	352	94	78	89	770	1333	77
79	83			2		2	2	1	1	11
22	19	11			1				2	1
1321	11		4			4				10
352	17						2			
94	12	1		1			2		2	
78	2					1				4
89	7			1		1	3			
770	4			1						
1333	11	2		1	1				2	
77	1					6				4

84 / 152

non-XSS detections
labeled CWE-79

GPT-5.1 · F1 62.8%

	79	22	1321	352	94	78	89	770	1333	77
79	115	1		8	1					
22	1	33			1					
1321			34							
352	2	1		23						
94	4				15					
78						7				
89							13			
770								10		
1333	1								4	
77										10

8 / 159

for GPT-5.1, but every
CWE-77 case was
called CWE-78

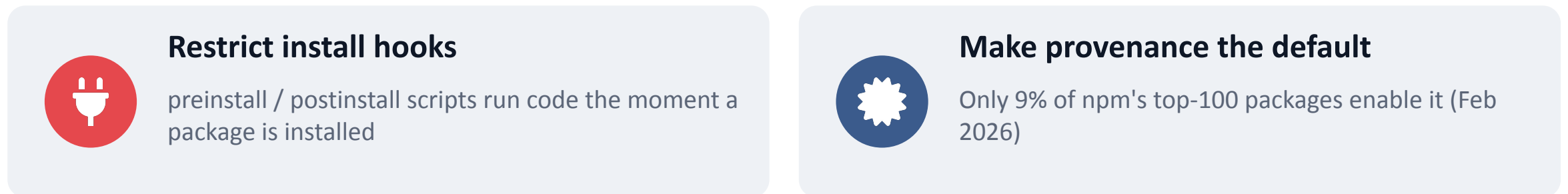
True-positive detections only. Rows: actual CWE. Columns: predicted CWE (Fig. 5).

Close pairs stay hard for every model: CWE-77 vs. 78 (command injection), CWE-862 vs. 863 (authorization).

A layered gatekeeper, not a single model



REGISTRY-SIDE LEVERS



Takeaways

91.5% vs 26%

Vulnerable packages grow far faster than ecosystems, while fixes are stuck at ~1,650 / yr

6 CWEs → 50%+

Risk is concentrated; malicious packages make up 67% of npm reports

62.8% F1

No single LLM is enough: layer fast screening with precise validation

Authors: Seyed Ali Akhavani, Behzad Ousat, Selcuk Uluagac, Amin Kharraz

github.com/fiu-seclab/oss-security

ali@akhavani.net



About me

Questions?

References

[1]

An Empirical Study on the Effectiveness of Static C Code Analyzers for Vulnerability Detection

Stephan Lipp, Sebastian Banescu, and Alexander Pretschner

ISSTA 2022, pp. 544–555. doi:10.1145/3533767.3534380

[2]

Comparison and Evaluation on Static Application Security Testing (SAST) Tools for Java

Kaixuan Li, Sen Chen, Lingling Fan, Ruitao Feng, Han Liu, Chengwei Liu, Yang Liu, and Yixiang Chen

ESEC/FSE 2023, pp. 921–933. doi:10.1145/3611643.3616262

[3]

Open Source, Open Threats? Investigating Security Challenges in Open-Source Software

Seyed Ali Akhavani, Behzad Ousat, and Amin Kharraz

arXiv:2506.12995, 2025. Earlier version of this research.