DistriNet KU LEUVEN

Origin Story

A Comprehensive Lifecycle Analysis of Same-Origin Policy Bugs

Jakub Szymsza · **Gertjan Franken** · Vik Vanderlinden · Tom Van Goethem · Mathy Vanhoef · Lieven Desmet

DistriNet, KU Leuven

ACM SecDev '26

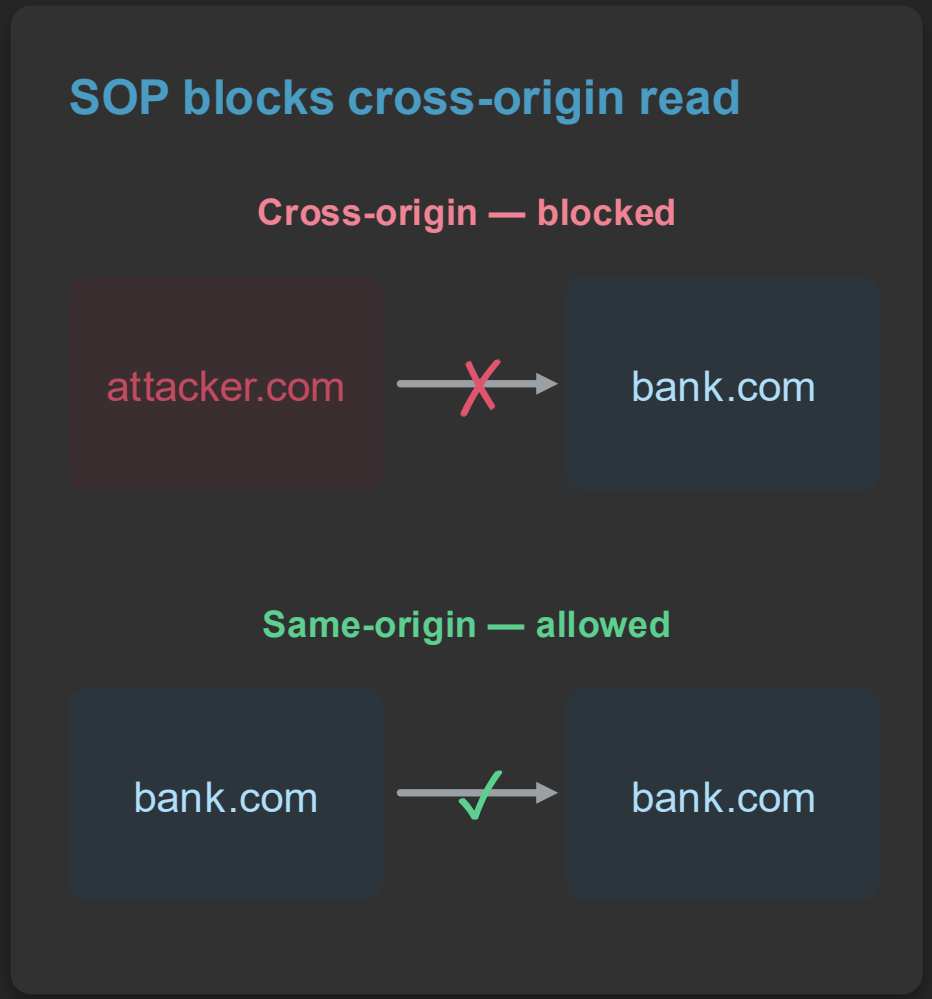
BACKGROUND

The Same-Origin Policy

- Introduced in Netscape Navigator in 1995
- One of the oldest, most fundamental web security policies
- Defense against cross-origin data theft
- Still essential defense in every modern browser

attacker.com | bank.com

```
// attacker.com: steal the victim's bank data
fetch('https://bank.com/api/balance', {
  credentials: 'include'   // ride session cookies
}).then(r => r.json())
  .then(data => exfiltrate(data));
```



MOTIVATION

The SOP has no unified specification

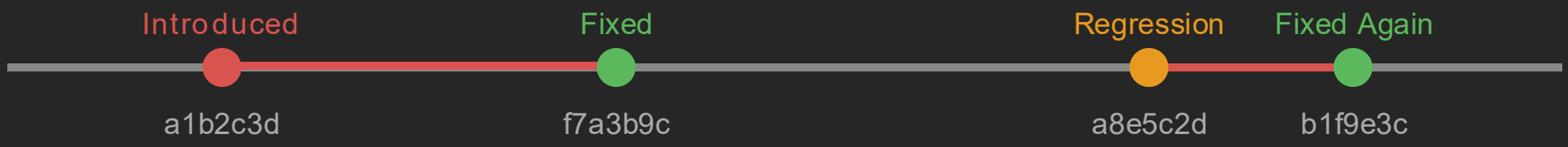
- Per the **W3C**: *“there is no single same-origin policy.”*
  Some parts evolved de facto over decades of independent vendor implementational choices.
- The SOP can be considered a **collection of policies.**
  Governs DOM access, fetch / XHR, storage, cookies, media, postMessage, etc.
- In this research we use the **broadest definition.**
  “all security policies that enforce access control or isolation boundaries based on a resource’s origin.”

No specification + very broad scope → fragile security policy

MOTIVATION

Plenty of SOP bugs

- Ever increasing amount of new web features relentlessly enlarges the attack surface
- This is compounded by the fragile nature of the SOP
- Still, we don't know much about SOP bug lifecycles
  No study has systematically analyzed how SOP bugs are introduced and resolved over time



Main research question

How are Same-Origin Policy bugs introduced and resolved across their full lifecycle?

METHODOLOGY

BugHog: tracing a bug's full lifecycle

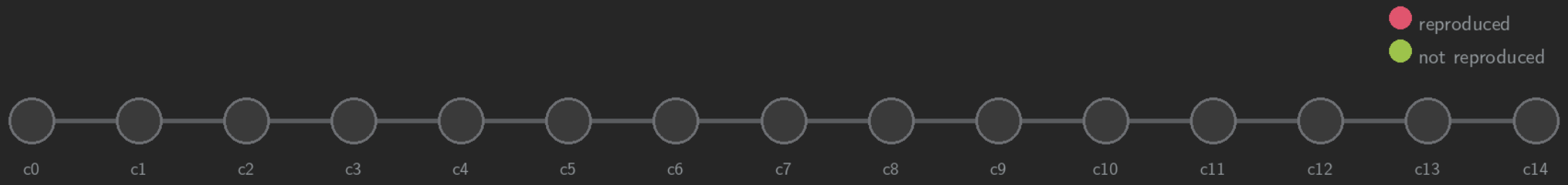
PRIOR WORK

A Bug's Life: Analyzing the lifecycle and mitigation process of Content Security Policy bugs

USENIX Security 2023

https://github.com/DistriNet/BugHog

- Open-source framework that automates revision-level bisection of browser builds
- Input: a bug's PoC, replayed across historical browser revisions
- Automatically fetches and instructs browser revision executables, and interprets when PoC is reproduced



METHODOLOGY

Extending BugHog for complex SOP PoCs

Complex user simulation

- Drives mouse & keyboard at the OS level (PyAutoGUI)
- Avoids WebDriver/CDP: unsupported on old builds

Dynamic server responses

- Custom per-request logic on the backend
- Responds based on request method & headers

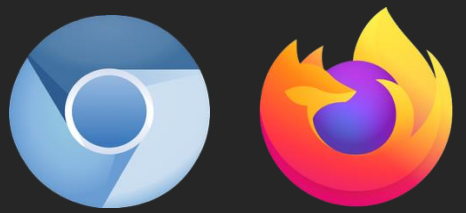
Other improvements

- Bug fixes, QoL improvements, etc

All extensions have been merged with the public BugHog open-source project

DATASET

From 242 reports to 97 analyzed bugs



242

candidate reports

keyword search of
Chromium & Firefox trackers

›

104

confirmed SOP bugs

after manual filtering
of false positives

›

102

reproduced

rebuilt as working PoCs

›

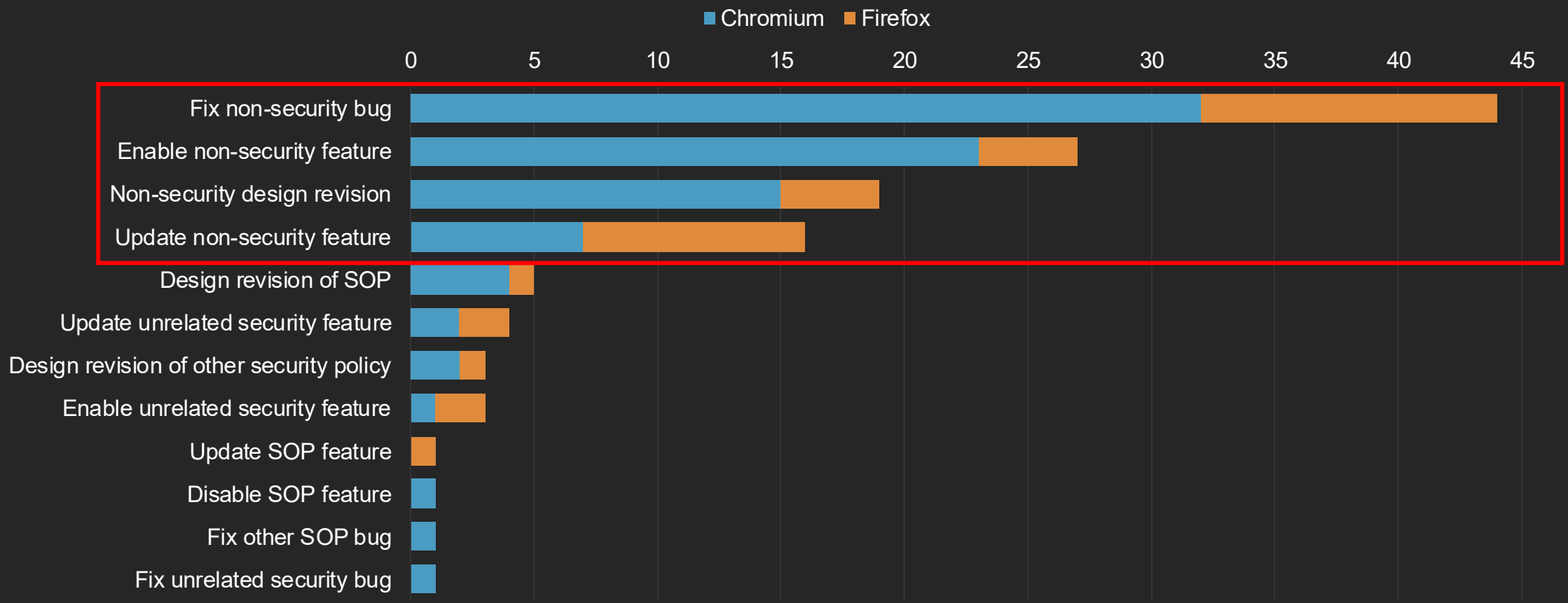
97

full lifecycle analyses

FINDINGS

What introduces SOP bugs

Top developer intentions behind the 125 introducing revisions (89 Chromium, 36 Firefox).



FINDINGS

Majority of SOP bugs are introduced by non-security changes

94%

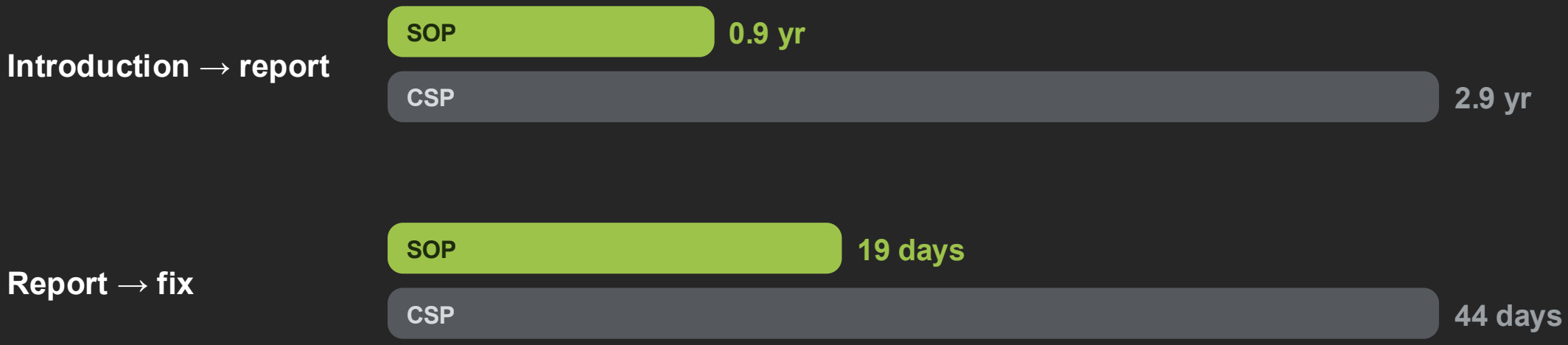
of the 125 introducing revisions had an intent unrelated to the SOP

- **The introducing revision's intent was unrelated to the SOP or security**
- **Sharp contrast with CSP**, where bugs were mostly introduced by changes to the policy logic itself
- **Implication:** heuristics that flag "security-related" commits could work for CSP, but would miss almost all SOP bugs

FINDINGS

SOP bugs live shorter lives than CSP bugs

Median Chromium lifecycle intervals: **SOP** vs the earlier **CSP** study



Underlying factors

› The SOP is considered a fundamental, always-on security boundary
› The CSP is considered defense in depth

FINDINGS

Bug labeling is inconsistent

19%

of SOP bugs correctly categorized in Chromium components

25% **19%**

of titles contained SOP keywords

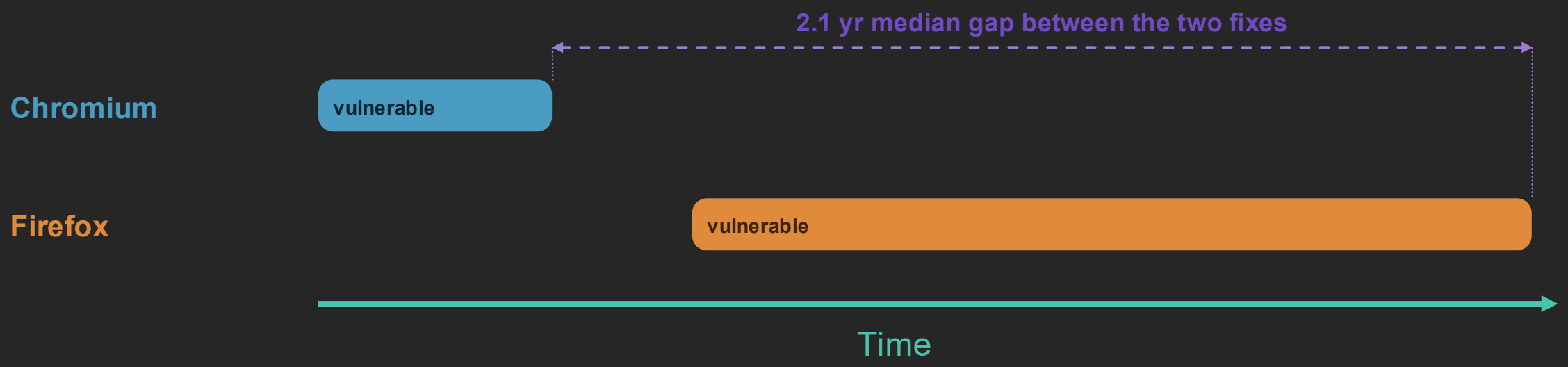
7% **0%**

of reports link the introducing revision

FINDINGS

Cross-browser bugs open long exploitation windows

- 23 of 97 bugs reproduced in both browsers
- Only 8 of the 23 had reports filed in both browsers



Public bug reports of one can be abused to reproduce unfixed bugs in the other.

FINDINGS

Three vulnerabilities still lived in the latest browsers

Discovered and responsibly disclosed, yet each vendor responded differently to the same flaw.

Status leak via <video> media errors

Leaks a cross-origin resource's 2xx status — e.g. login detection.


"intended"


original report


unresolved

Status leak via <script> events

Cross-origin status inferred from a <script> tag's load vs error events.


original report


patched


"intended"

Origin header not cleared on redirect

Redirected POST keeps the original Origin → CSRF against a trusted origin.


original report


fixing


no bug

Recommendations

1 Build SOP checks into every feature

Treat SOP compliance as an essential part of design and regression testing for any new mechanism.

2 Standardized and consistent bug labeling

A shared taxonomy across vendors so SOP issues are findable and comparable.

3 Pragmatically formalize the SOP

Don't specify everything, instead prioritize reconciling behaviors where browsers disagree today.

CONCLUSION

Recap & what's next

The first comprehensive lifecycle analysis of SOP bugs: 97 across Chromium & Firefox

KEY TAKEAWAYS

- **94% of SOP bugs come from non-security changes**
  The opposite of CSP, so security-bug models don't generalize across policies.
- **Three cross-vendor vulnerabilities were still live**
  Vendors don't even agree on what counts as an SOP bug.
- **Standardization is the path forward**
  Consistent bug labeling and reconciling divergent vendor behavior.

BugHog source code

What's next

- Further automatization of bug lifecycle identification process
- Extending BugHog to more platforms