black hat USA 2020

AUGUST 5-6, 2020

BRIEFINGS

The Devil's in the Dependency: Data-Driven Software Composition Analysis

Ben Edwards
Cyentia Institute

Chris Eng
Veracode

We’re going to demonstrate, with data…

Even the smallest library (162 LoC) can introduce flaws into your application

Most libraries aren’t even directly included, but are included by other libraries – a blind spot for developers

More libraries doesn’t always mean more problems

There are better ways to prioritize fixes than by severity

Rejoice! 81% of patchable vulnerabilities can be fixed with a minor library update, and most updates are small – even when updates introduce new flaws!

About us

Ben

Senior Data Scientist @ Cyentia

PhD in CS applying data science to security

Wide gamut of published research: breaches, botnets, AI security, privacy, policy, and cyberwar

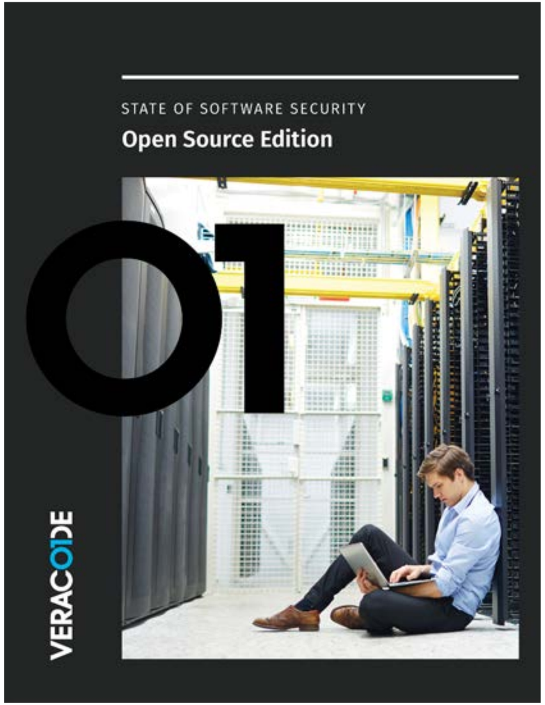
Chris

Chief Research Officer @ Veracode

20+ years in application security: build, break, and defend

Been involved with SoSS since Volume 1 (2010)

About the report



Veracode State of Software Security (SoSS), released annually-ish since 2010

Joint venture with Cyentia Institute since 2018

Motivations

- Insights into industry performance
- Customer benchmarking
- Actionable advice for improving AppSec

This talk includes additional research not covered in the original report!

Agenda

Data sources and biases

Library usage

Transitive dependencies

Flaw categories and patterns

Fix prioritization, evolved

Update chains

Data sources

Largest known quantitative study of application security findings

12 months of application scan data

Over 85,000 unique applications and 351,000 unique libraries

Biases

Experimental errors: Type I (false positives) and Type II (false negatives)

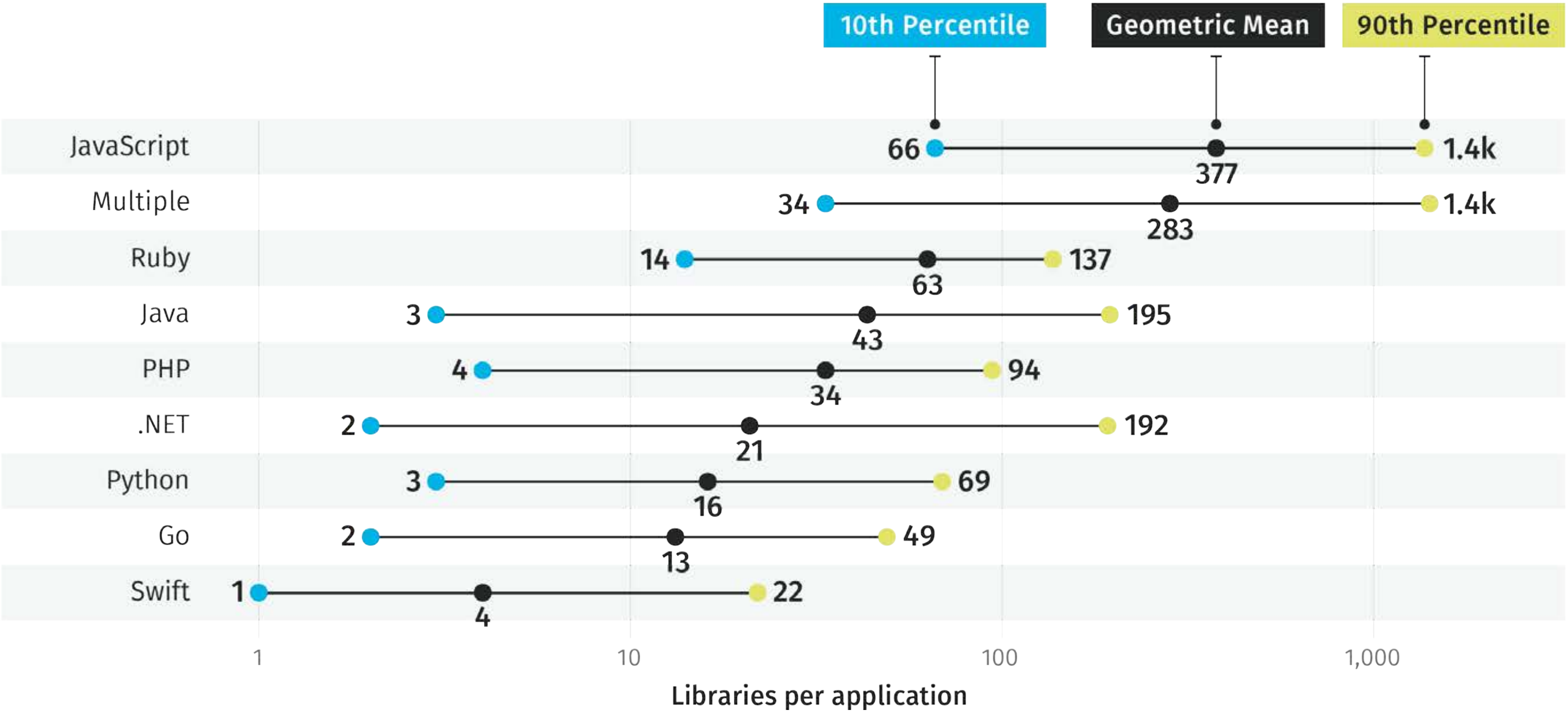
Selection bias, e.g. who are Veracode's customers, which applications did they choose to analyze, etc.

Attribution bias, e.g. inclination to "blame" outcomes on things that seem relevant (e.g. developer skill) vs. other situational factors (e.g. release deadlines)

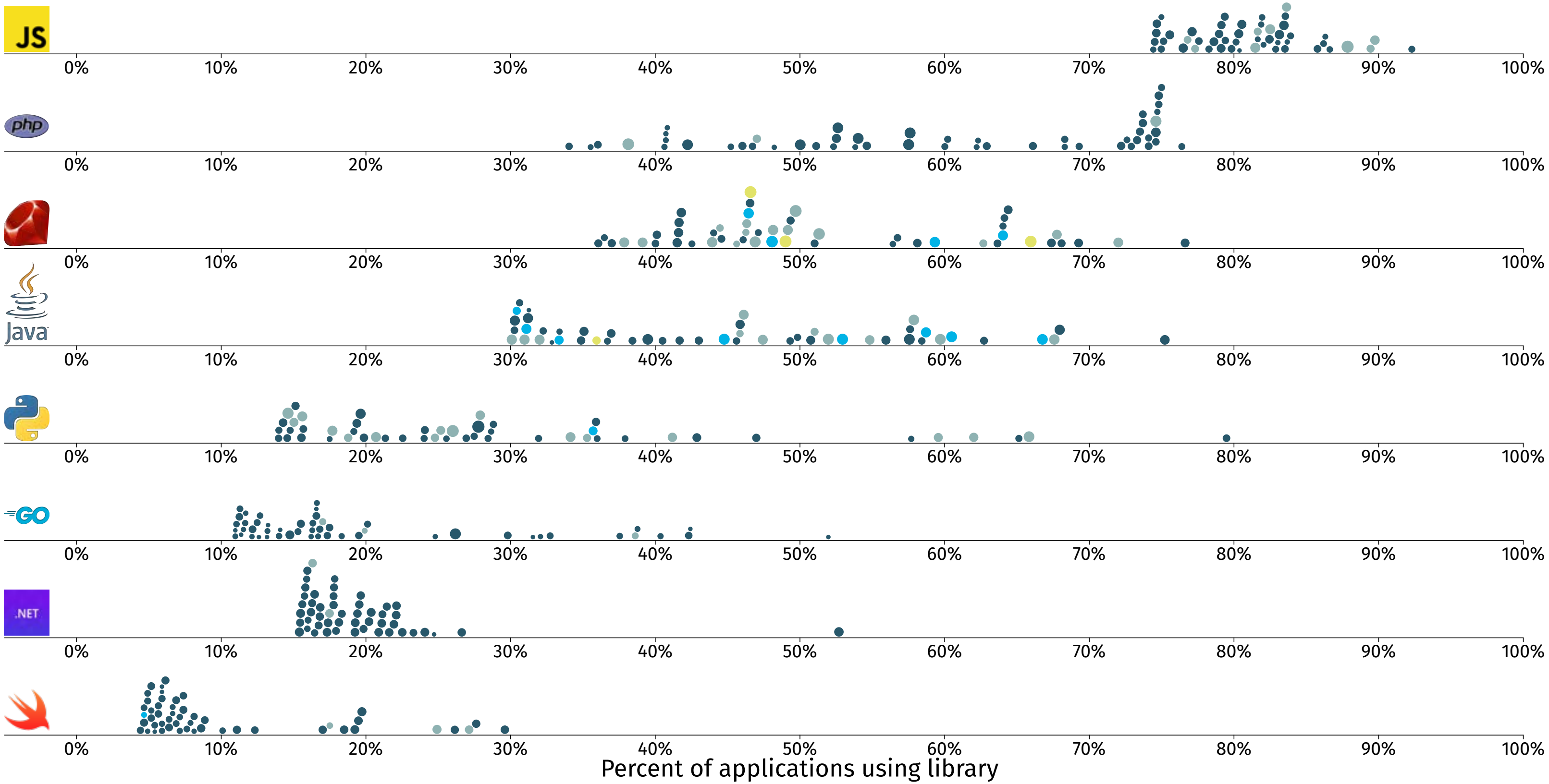
Library usage

The Open Source popularity contest

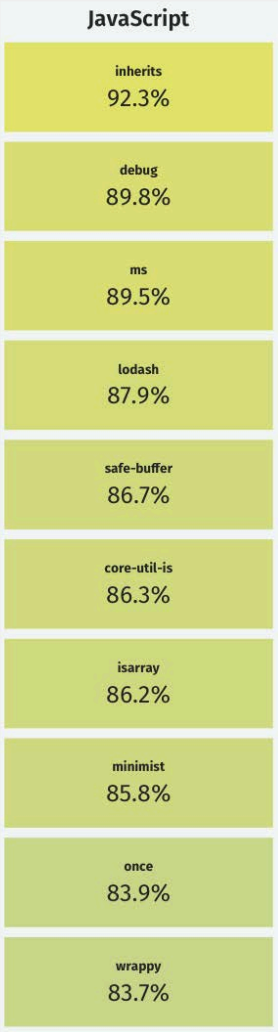
Library usage is highly language dependent



Usage rate of popular libraries

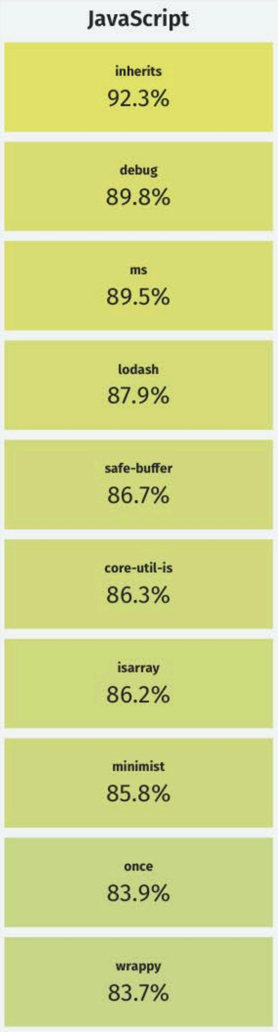


We need to talk about JavaScript...



We need to talk about JavaScript Top 10 libraries

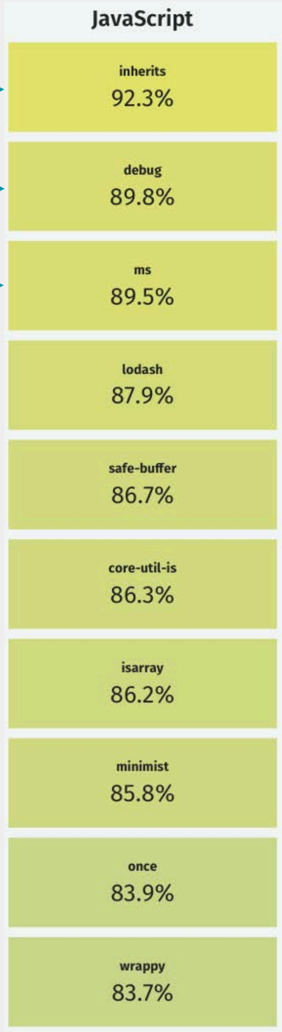
Incredibly numerous and small libraries



We need to talk about JavaScript Top 10 libraries

Incredibly numerous and small libraries

Top 3 have < 1 kLOC each (36, 790, 162 respectively)

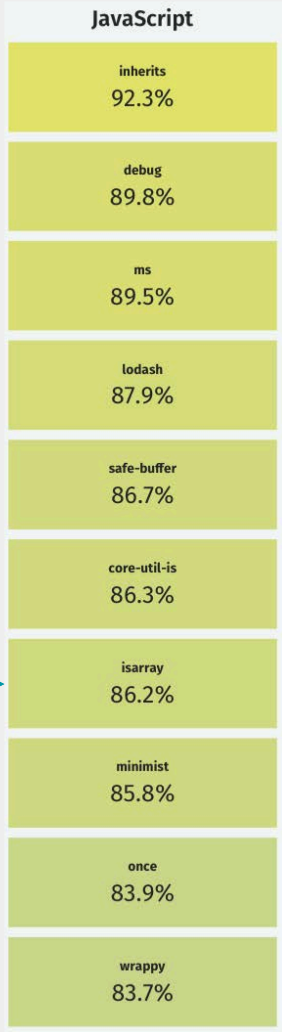


We need to talk about JavaScript Top 10 libraries

Incredibly numerous and small libraries

Top 3 have < 1 kLOC each (36, 790, 162 respectively)

`isarray` is only 4 lines long



@benjamesedwards @chriseng

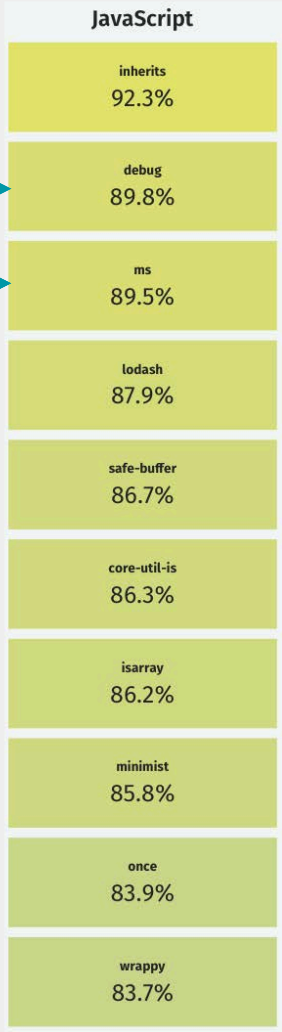
We need to talk about JavaScript Top 10 libraries

Incredibly numerous and small libraries

Top 3 have < 1 kLOC each (36, 790, 162 respectively)

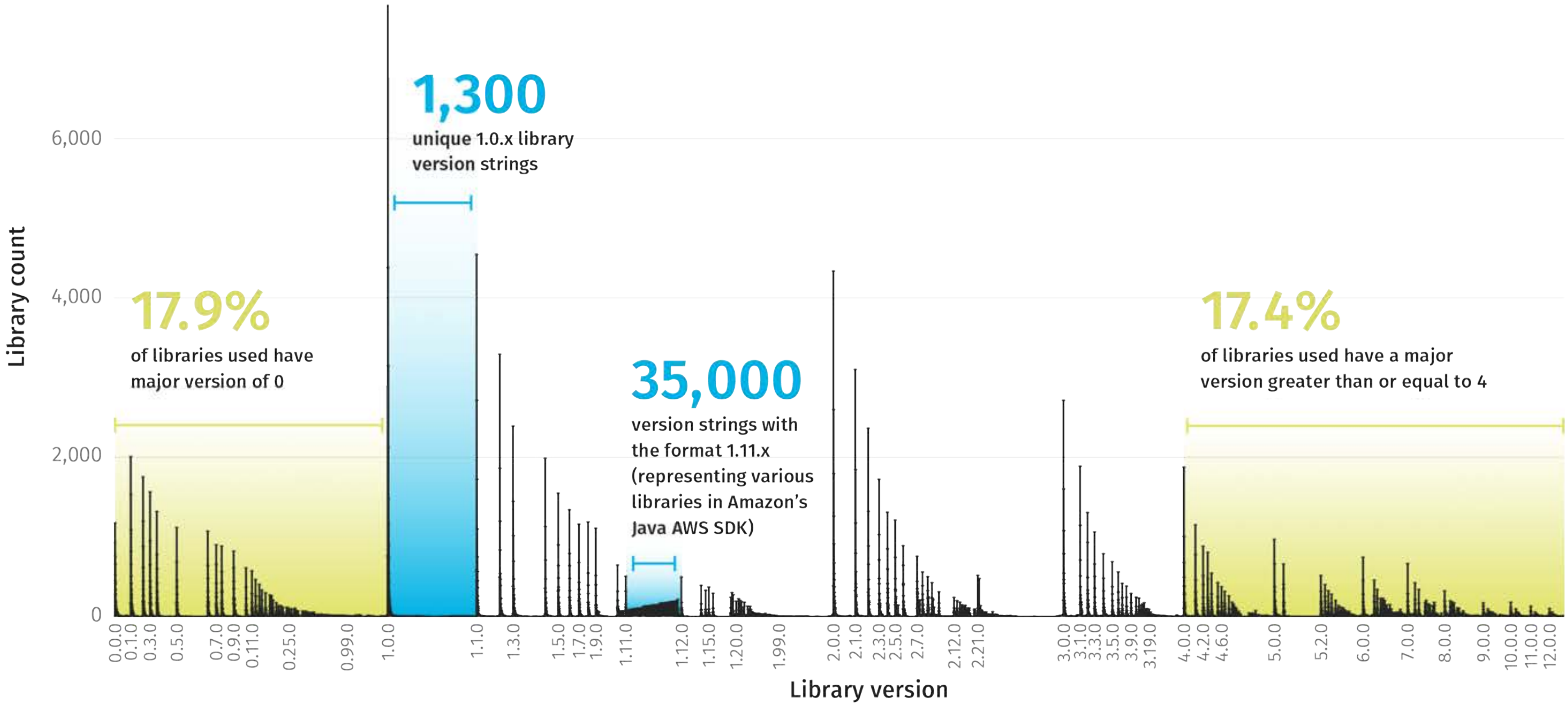
`isarray` is only 4 lines long

`debug` and `ms` have versions with CVEs



@benjamesedwards @chriseng

SemVer, the closest we can get to a standard...



Transitive dependencies

It's libraries all the way down...

Definition / implications

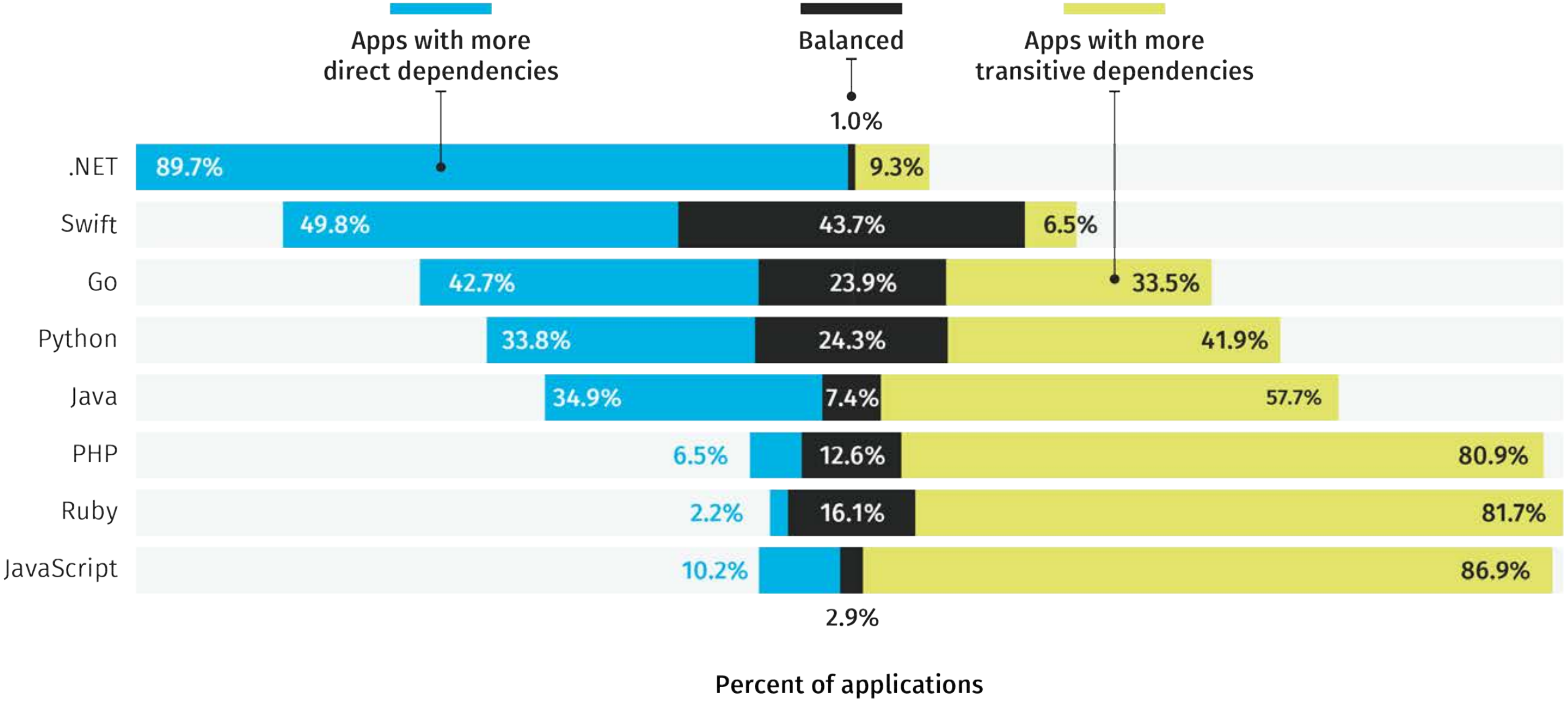
Libraries, like applications, aren’t built in a vacuum

Including a library means including every library it uses

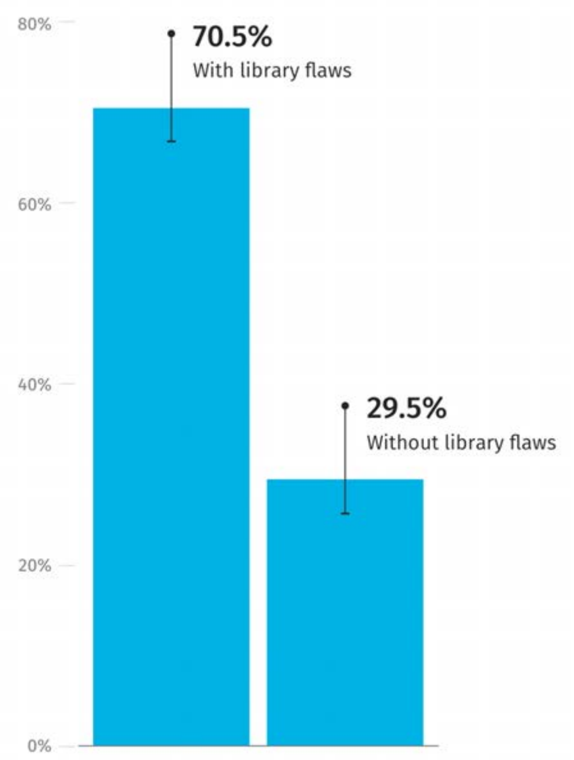
Two types of dependencies

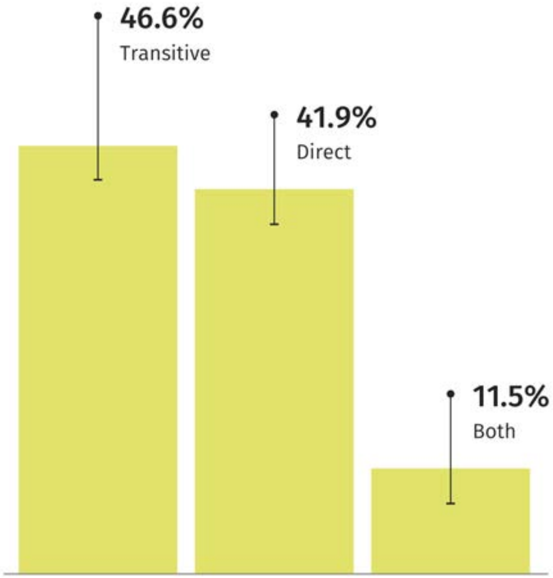
- **Direct Libraries** that are explicitly included by the developer
- **Transitive Libraries** that are included by another library

Transitive by language (Fig 4)



Direct vs Transitive vulnerabilities (Figs 11-12)

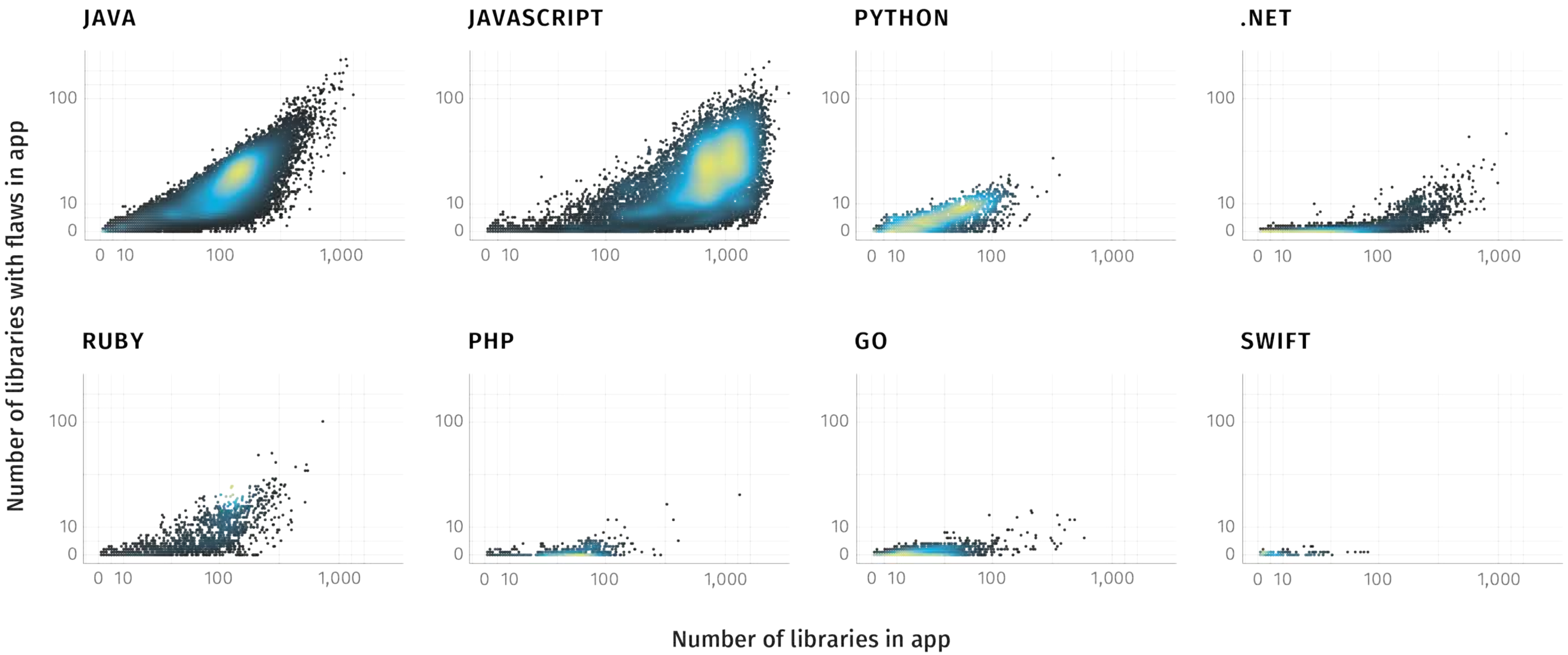




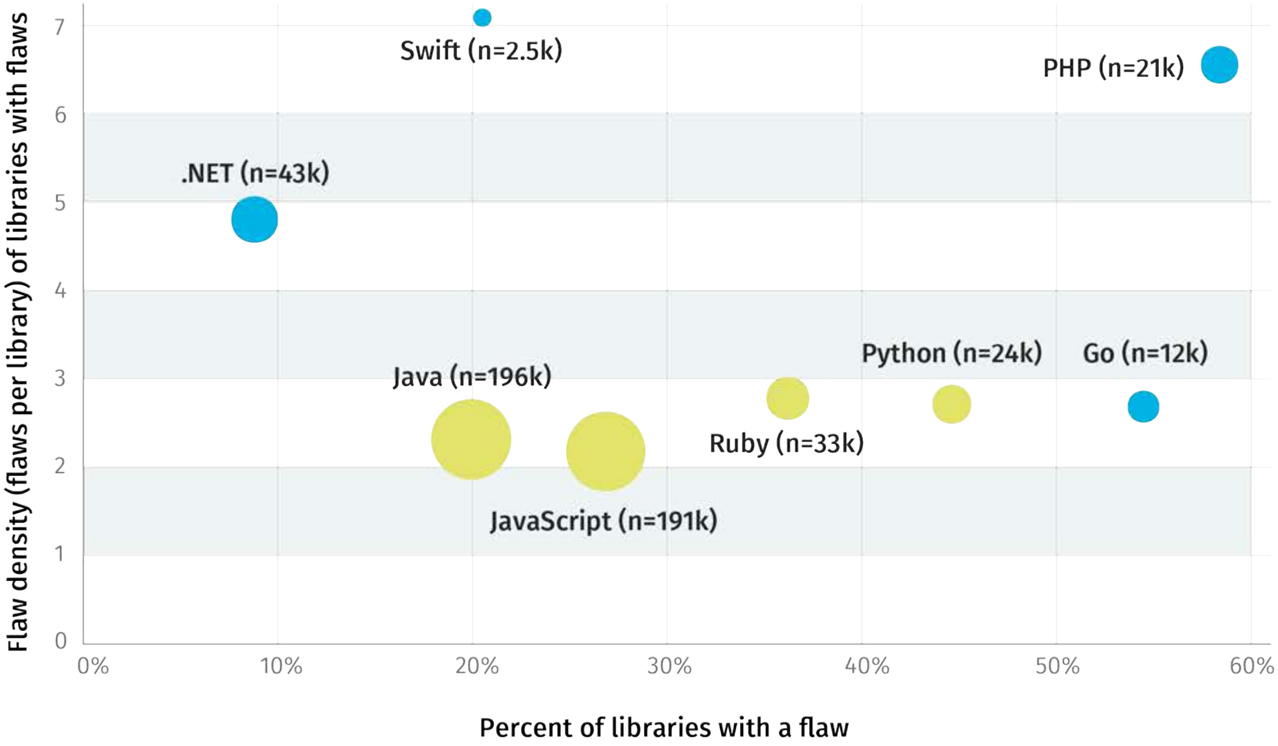
Flaws

Every rose has its thorn…

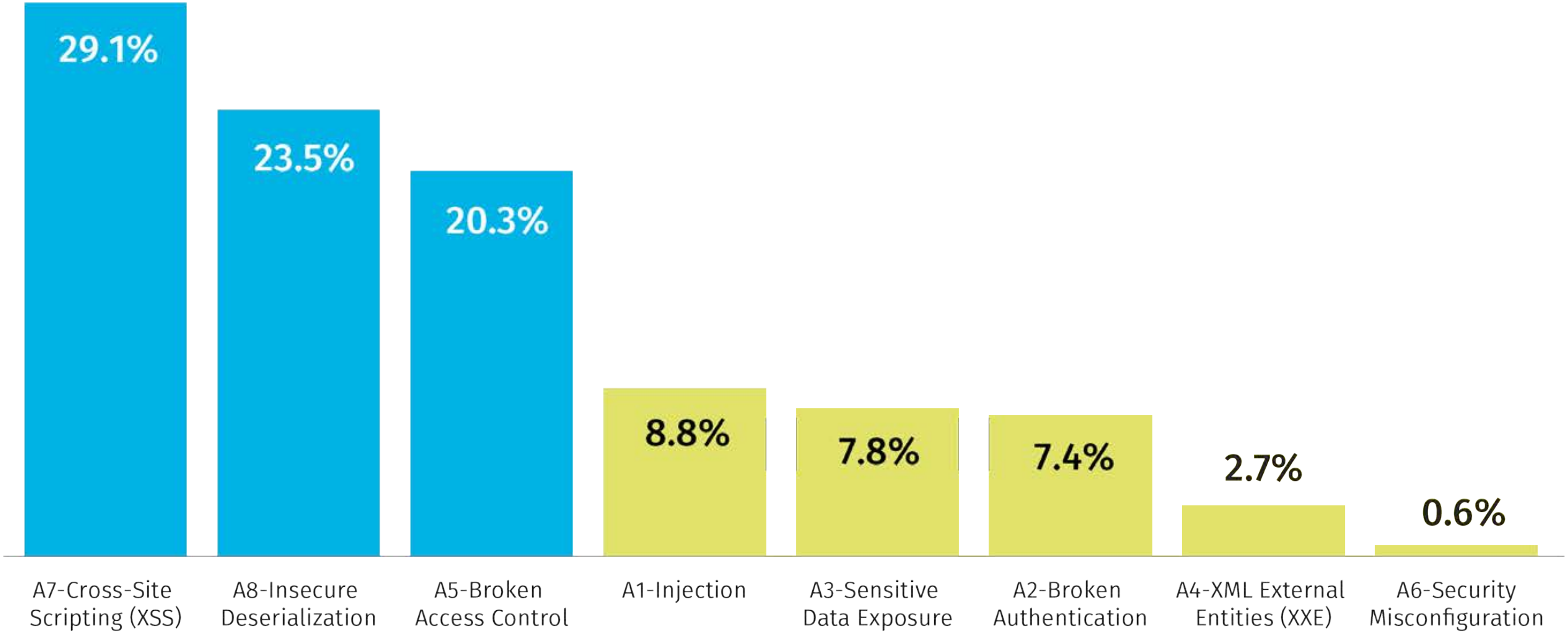
More libraries = more problems? (Fig 13)



Language choice makes a difference (Fig 5)



OWASP Top Ten (Fig 6)



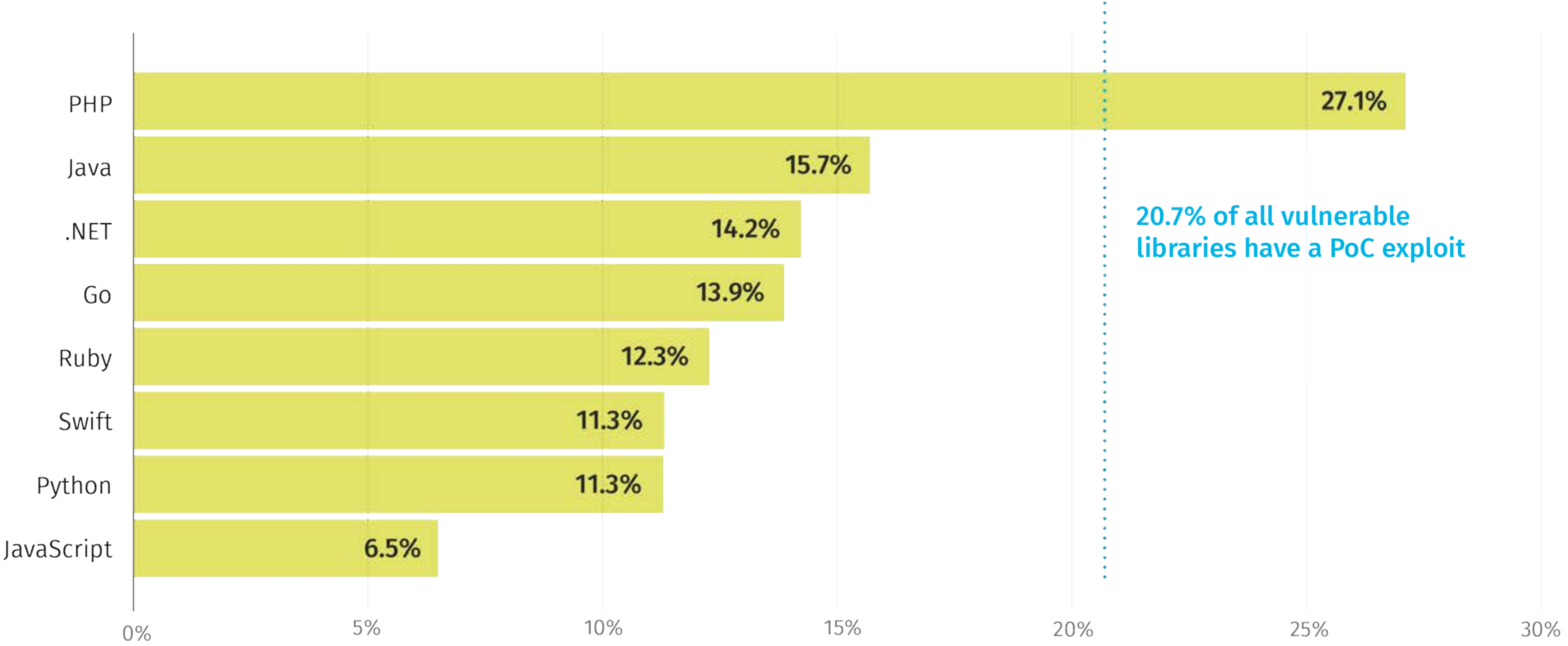
PHP is basically a minefield (Fig 7)

	Go	Java	JavaScript	.NET	PHP	Python	Ruby	Swift
A1-Injection	3.4%	1.7%	2.5%	2.9%	18.6%	6.3%	7.8%	0.0%
A2-Broken Authentication	4.9%	6.9%	1.9%	1.9%	21.3%	6.5%	3.2%	0.2%
A3-Sensitive Data Exposure	8.0%	2.1%	0.6%	8.8%	4.6%	2.6%	1.4%	6.1%
A4-XML External Entities (XXE)	0.0%	5.9%	0.0%	0.5%	0.1%	1.6%	0.5%	0.2%
★ A5-Broken Access Control	10.7%	8.9%	4.9%	14.8%	22.5%	9.4%	8.0%	7.7%
A6-Security Misconfiguration	0.0%	0.7%	0.2%	0.0%	1.2%	0.0%	0.0%	0.0%
★ A7-Cross-Site Scripting (XSS)	11.0%	10.5%	11.6%	8.4%	40.1%	13.3%	13.9%	0.0%
★ A8-Insecure Deserialization	0.0%	7.6%	0.0%	0.4%	17.4%	0.9%	1.5%	0.0%

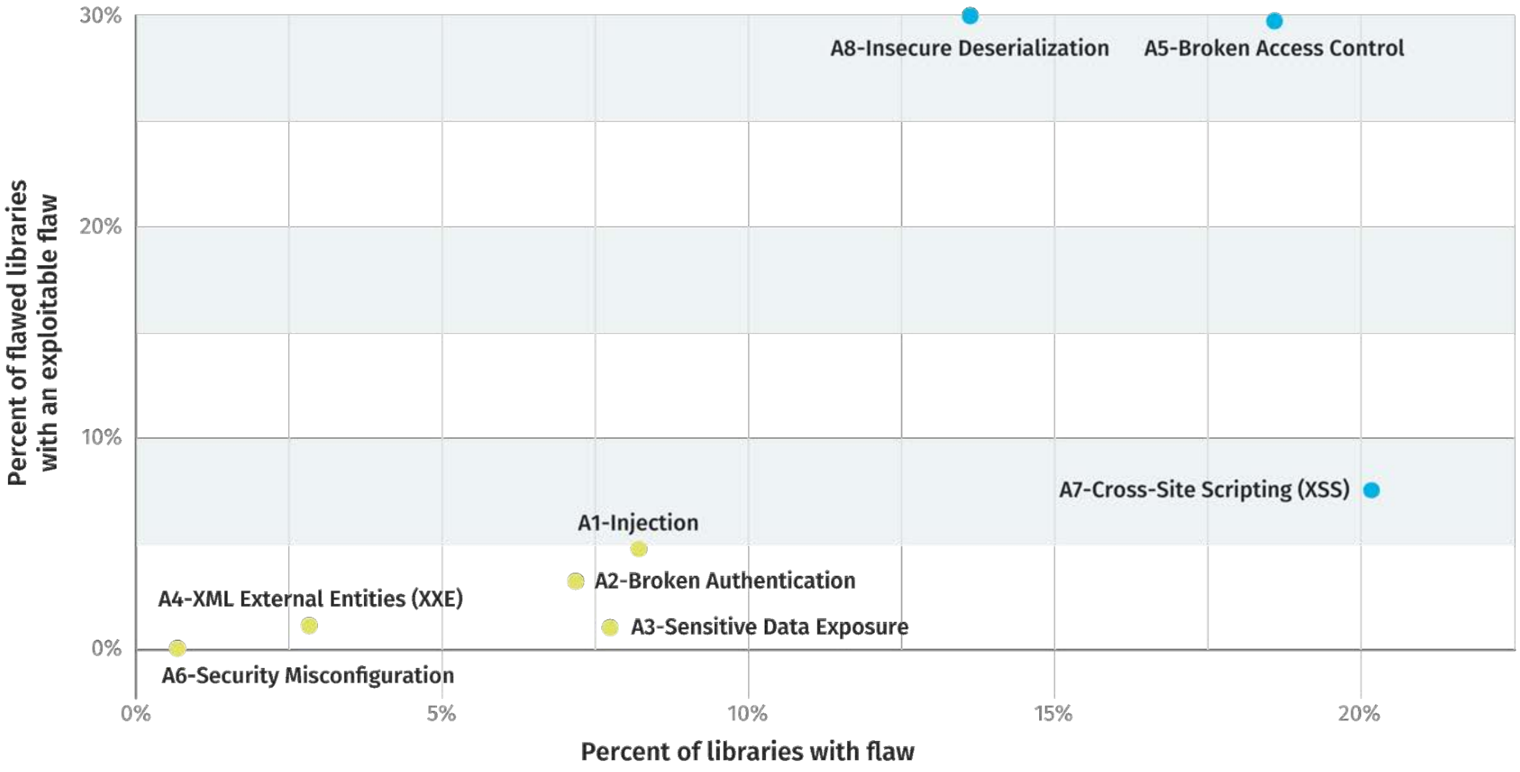
Alright, now what?

Prioritizing fixes

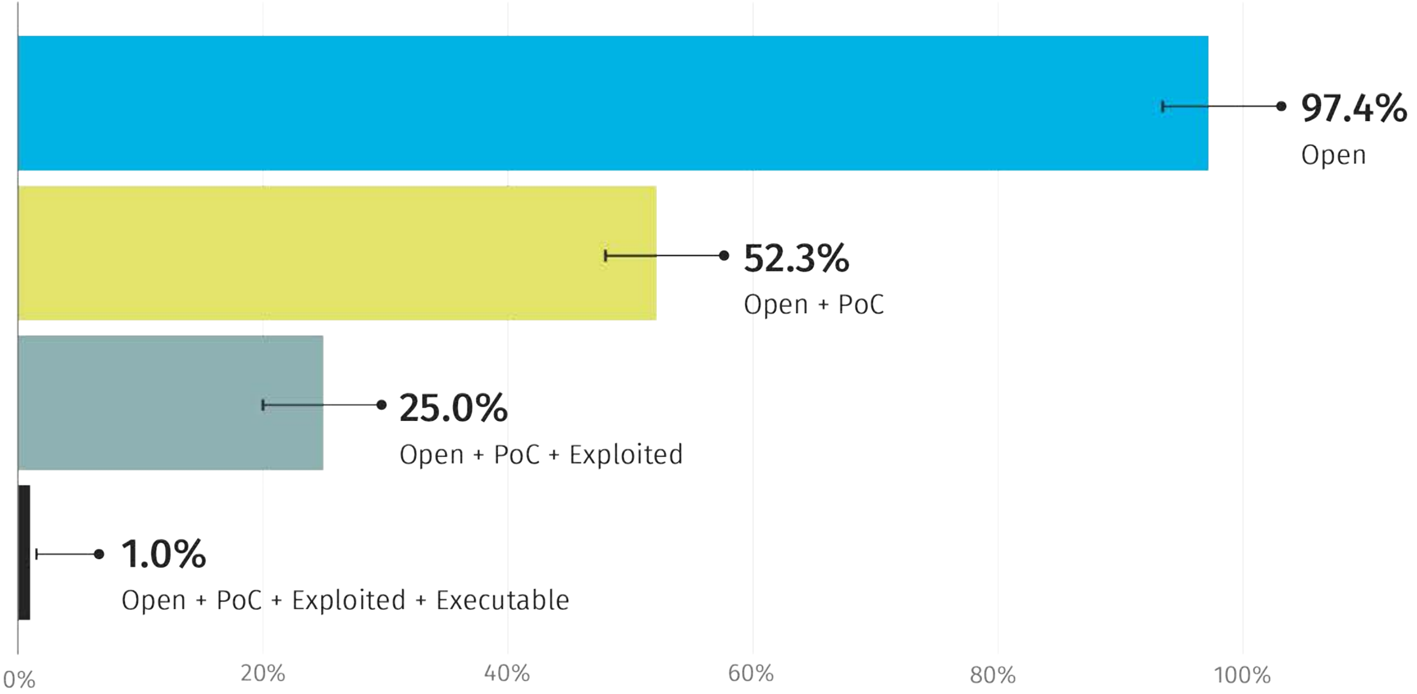
Not all vulnerabilities have exploits (Fig 8)



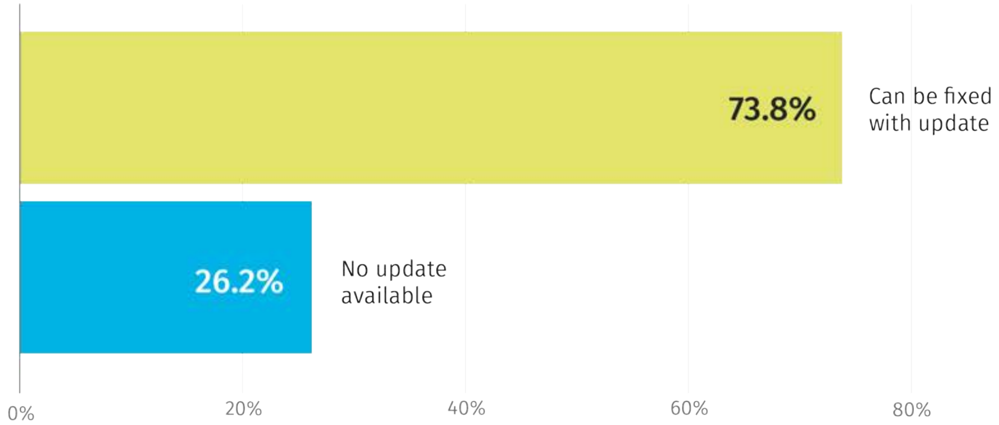
PoC exploits by OWASP category (Fig 10)

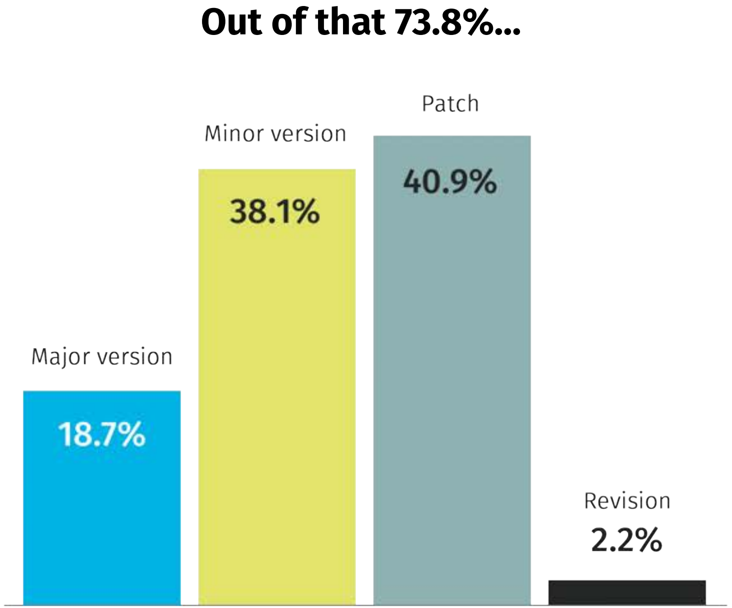


The vulnerability funnel (Fig 14)



Good news: most fixes are minor (Figs 16-17)





It’s never that easy…

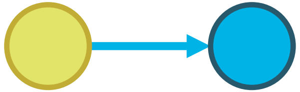
Updating a library can introduce new flaws, which require further updates, which may introduce new flaws, requiring more updates…

So what do these update chains look like?

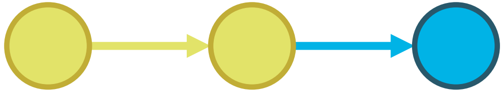
It’s never that easy...

Updating a library can introduce new flaws, which require further updates, which may introduce new flaws, requiring more updates...

So what do these update chains look like?



Single step to version
with no known flaw

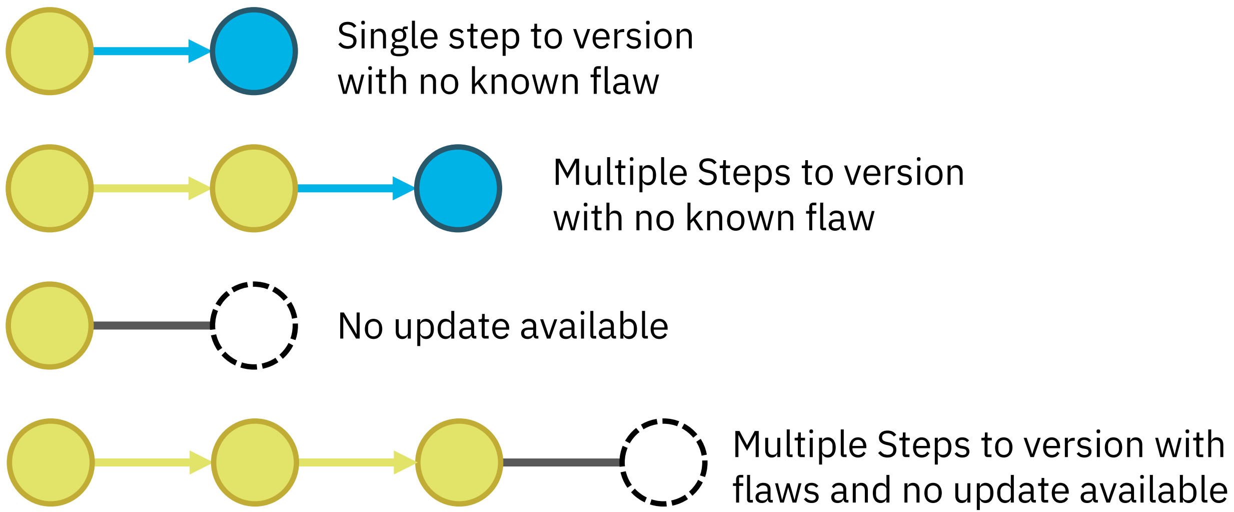


Multiple Steps to version
with no known flaw

It’s never that easy…

Updating a library can introduce new flaws, which require further updates, which may introduce new flaws, requiring more updates…

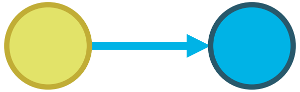
So what do these update chains look like?



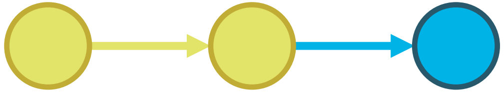
It’s never that easy…

Updating a library can introduce new flaws, which require further updates, which may introduce new flaws, requiring more updates…

So what do these update chains look like?

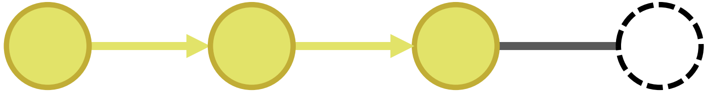


Single step to version with no known flaw

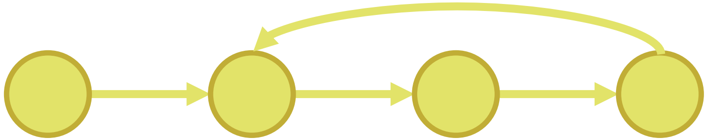


Multiple Steps to version with no known flaw

No update available



Multiple Steps to version with flaws and no update available



Suggested updates are circular

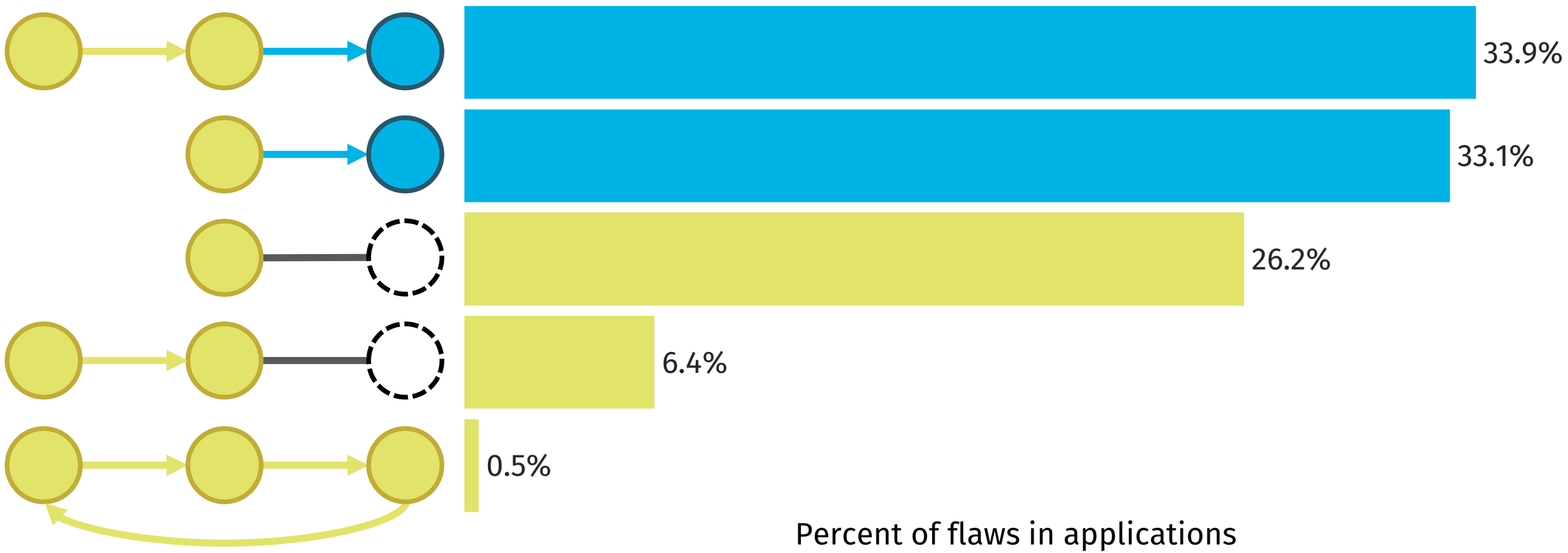
Begs many questions

How do these chains end?

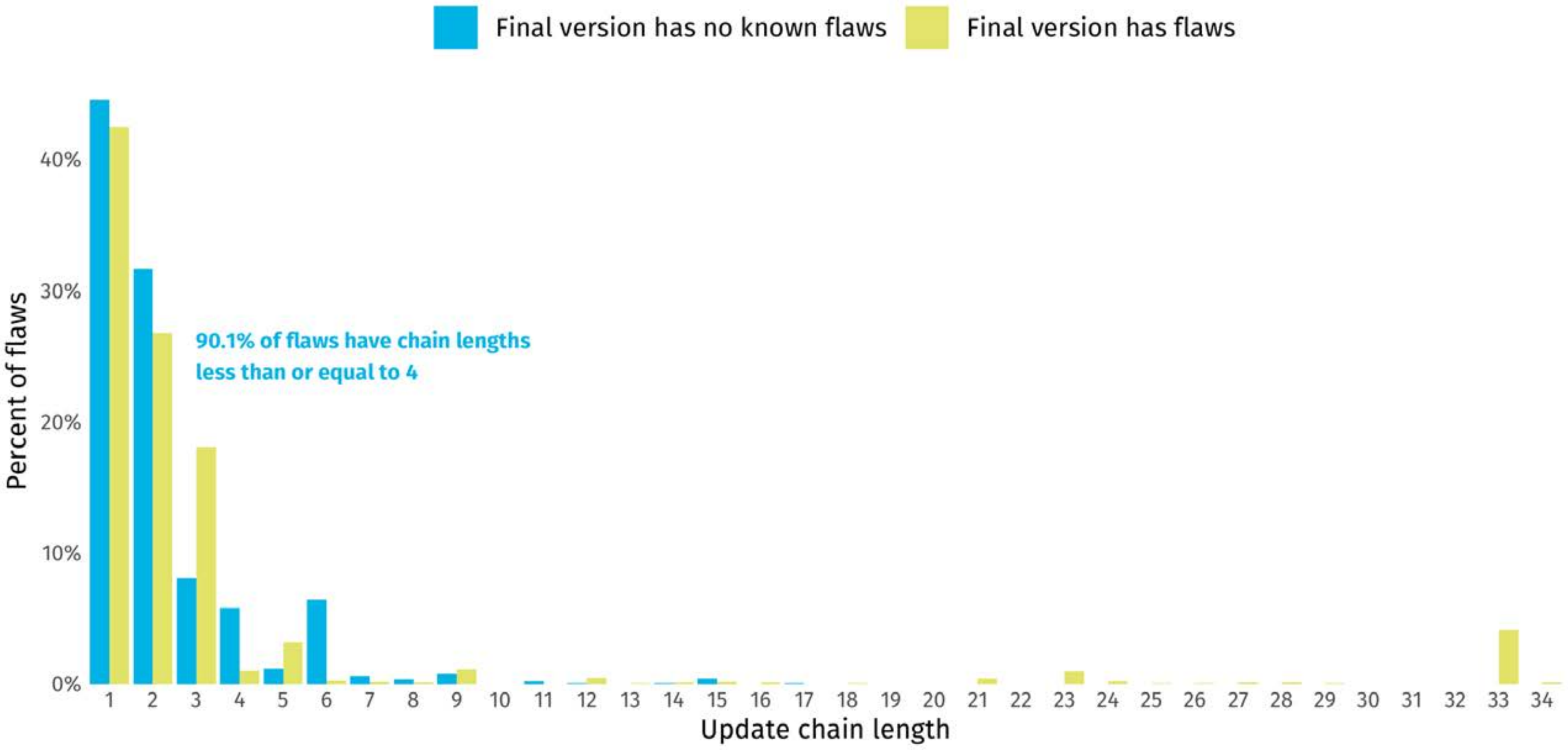
How many steps do they have?

Do they significantly increase update size?

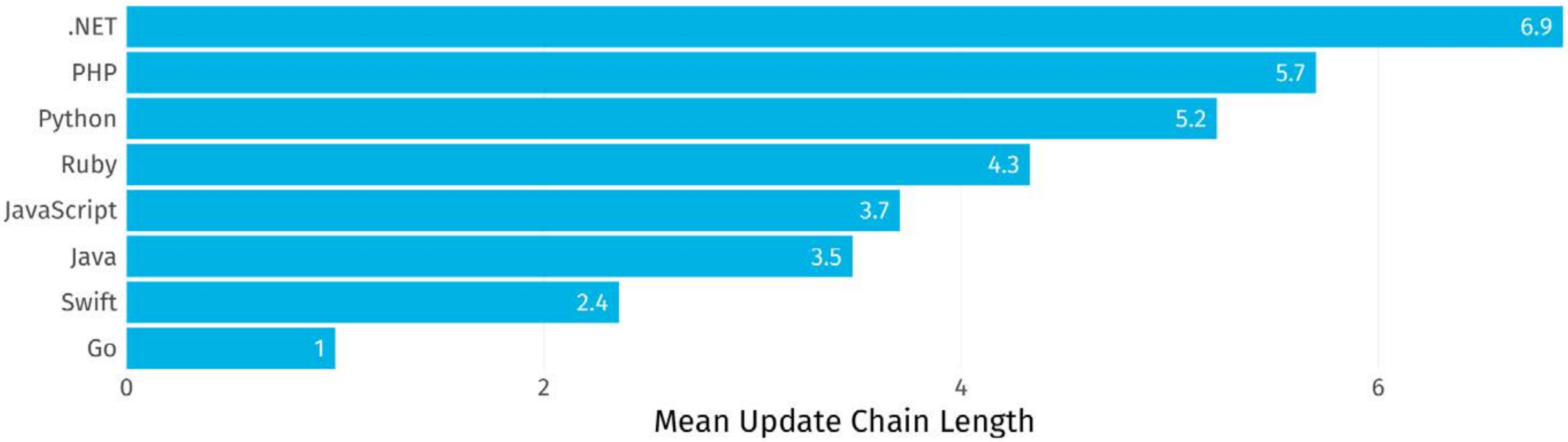
How do the chains end?



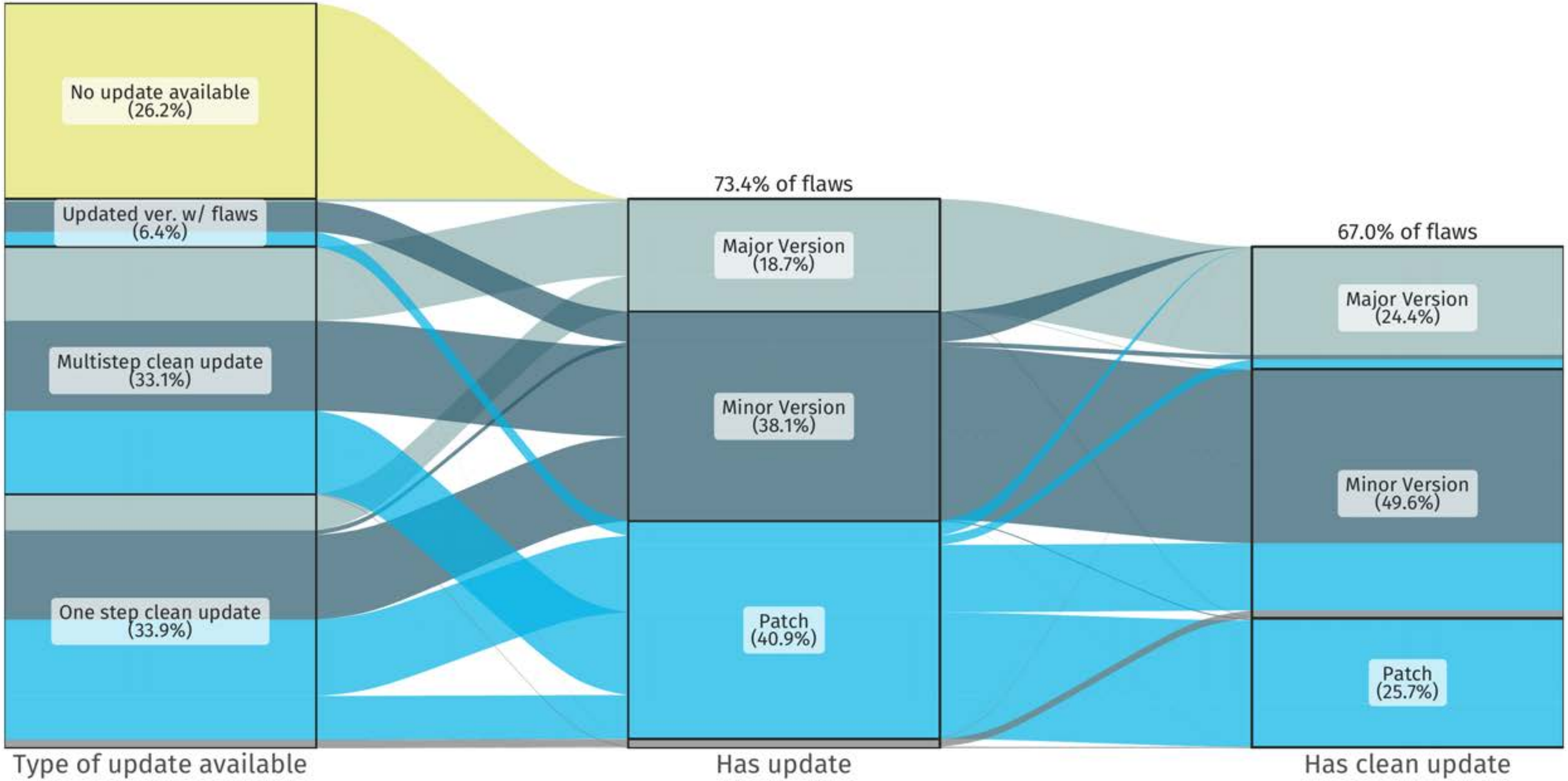
Most chains are relatively short...



… but it varies by language



Most updates are still small



Conclusions / Q&A

Takeaways

Open source software has a surprising, and surprisingly variable, number and type of software flaws.

The attack surface of many applications — due to the transitive dependency phenomenon — is much larger than developers may expect.

Language selection does make a difference — both in terms of the size of the ecosystem and in the prevalence of flaws in those ecosystems.

Most fixes are relatively minor in nature, suggesting that this problem is one of discovery and tracking, not huge refactoring of code.

Questions?

ben@cyentia.com

ceng@veracode.com