VERACODE

STATE OF SOFTWARE SECURITY: VOLUME 11

Open Source Edition

The Life and Times of Third-Party Software

https://t.me/learn

CONTENTS

SECTION ONE

02 **Introduction**

04 Key insights

SECTION TWO

05 **Popular Libraries**

SECTION THREE

09 **Library Selection**

10 Library selection process

11 Library evaluation and problems

13 Most libraries are *never* updated

15 How long do they stick around before being updated?

SECTION FOUR

16 **Fixing Vulnerabilities**

19 Severity

20 Dependency type

21 Vulnerability type

22 Language

23 Developer resources

SECTION FIVE

25 **Suggested Updates**

26 Most updates are *still* small

30 Update chains

SECTION SIX

33 **Conclusion**

34 Appendix: Methodology

https://t.me/learningnets

SECTION ONE

If you are reading this report, you probably already know that third-party software is the foundation of nearly all modern applications.

But that foundation is not a metaphorical solid block of cement on which to construct our software edifice. While Randall Monroe is rarely wrong, it may not even be the precarious pile of blocks conceived in one of his comics,[1] but perhaps a shifting pile of sand and gravel.

How does a developer navigate the shifting sands of the third-party library landscape? Last year, we looked at a snapshot of library usage in applications, covering how many libraries were used and what types of vulnerabilities were present in those libraries. We gave practical advice on what the fixes to those vulnerabilities looked like and had some heartening news — that if developers stay on top of updates, that will address most problems.

[1] *xkcd.com/2347*

https://t.me/learningnets

This year, again in partnership with the Cyentia Institute, we want to look beyond a point-in-time snapshot, examining the dynamics of library development and how developers react to library changes, including the discovery of flaws. We'll look at how library popularity has changed over the last two years. We'll see how often and quickly libraries are updated (not often), and how quickly flaws are addressed (surprisingly quickly!). We'll also dive deeper into those fixes and see when updates might be hard.

We also turned our attention not just to the data on software components, but to the voice of developers themselves. We conducted a survey of Veracode users to better understand their development practices and how they utilize third-party software. We then matched up survey responses to our technical data and can show how developer priorities can affect their application's security.

We'll see that reacting to the shifting landscape requires the right priorities with the right information, and with both, making applications more secure is not overly taxing. We hope the insights within can help developers more effectively utilize third-party software.

READ ON TO LEARN MORE ABOUT THE LIFE AND TIMES OF THIRD-PARTY LIBRARIES.

https://t.me/learningnets

KEY INSIGHTS

Open source libraries are constantly evolving; what appears secure today may not be tomorrow.

Despite this dynamic landscape, 79 percent of the time, developers never update third-party libraries after including them in a codebase.

79%

When alerted to vulnerable libraries, developers can act quickly.

17% of flaws are addressed within ONE HOUR

25% of flaws are addressed within ONE WEEK

65%

Sometimes updates beget more updates, but even when they do, 65 percent of those updates are a minor version change or smaller, and therefore unlikely to break the functionality of even the most complex application.

92% of library flaws can be fixed with an update

69% of updates are a minor version change or less

When developers understand the implications of vulnerabilities and appropriately prioritize security, they can fix most flaws easily.

Lack of contextual information, for instance about how a vulnerable library relates to their application, can be a roadblock for developers.

DEVELOPERS WHO LACK THE INFORMATION THEY NEED, TAKE:

7+ months to fix 50% of flaws

DEVELOPERS WHO HAVE THE INFORMATION THEY NEED, TAKE:

3 weeks to fix 50% of flaws

https://t.me/learningnets

SECTION TWO

POPULAR LIBRARIES

Before we examine the life and times of any given library, we should examine which libraries are currently living large.

https://t.me/learningnets

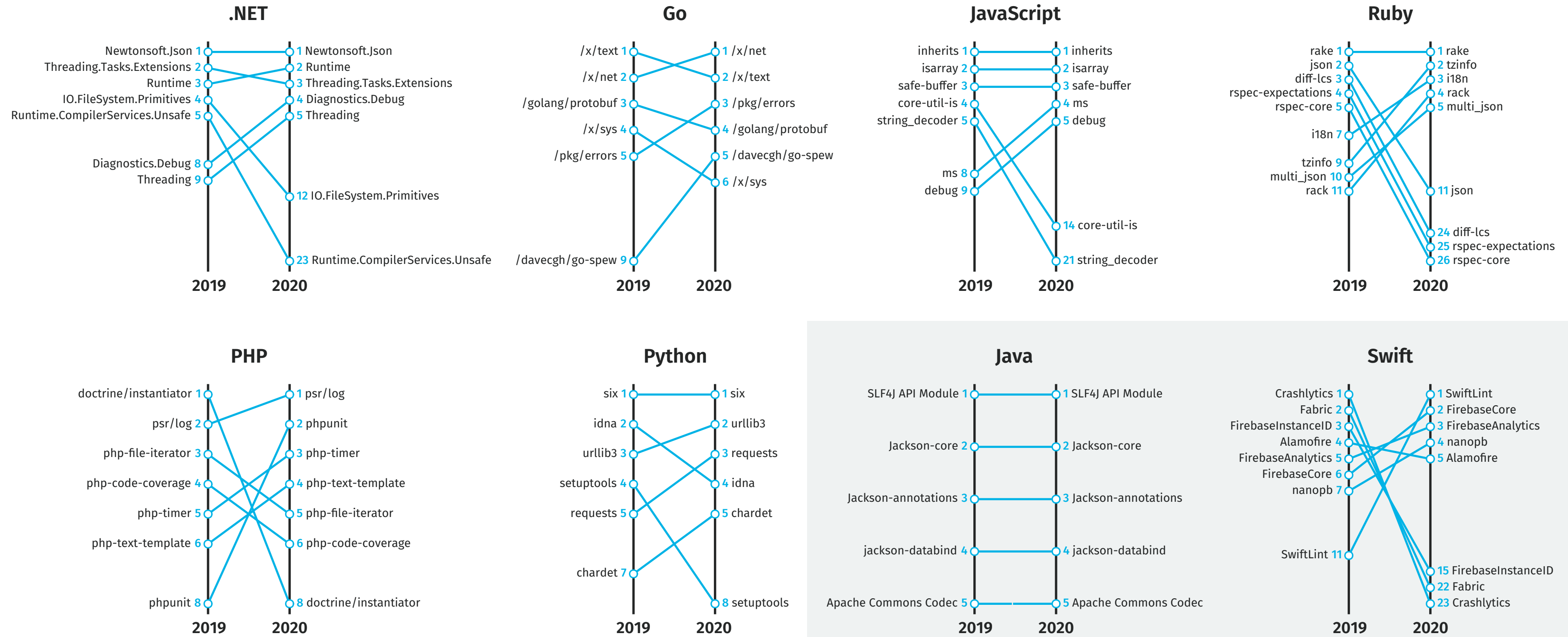


Figure 1 *Top libraries from 2019 and 2020*

Last year, we looked at the top ten[2] most popular libraries (by name) across a number of different languages.

We could do that again, but looking at the overall popularity with one more year of data isn't going to move things around too much. So this year, we examine the year-over-year popularity of libraries in Figure 1. We looked at all the libraries that made an appearance in the top five (by percentage of applications using the library) in either 2019 or 2020 and looked at how their relative ranks changed.

[2] *Or top 50 if you want to check out this interactive.*

JAVA

For some languages, there is little change. Java, with a long-running and robust third-party library ecosystem, sees no change in the top five.

SWIFT

In contrast, Swift looks like a shaken snow globe, with the top two libraries from 2019, Crashlytics and Fabric, not even breaking the top 20 in 2020. The reason is simple: Google (the parent company behind Firebase) acquired both companies and integrated the functionality into Firebase, leading to the meteoric rise in two Firebase libraries.

https://t.me/learningnets

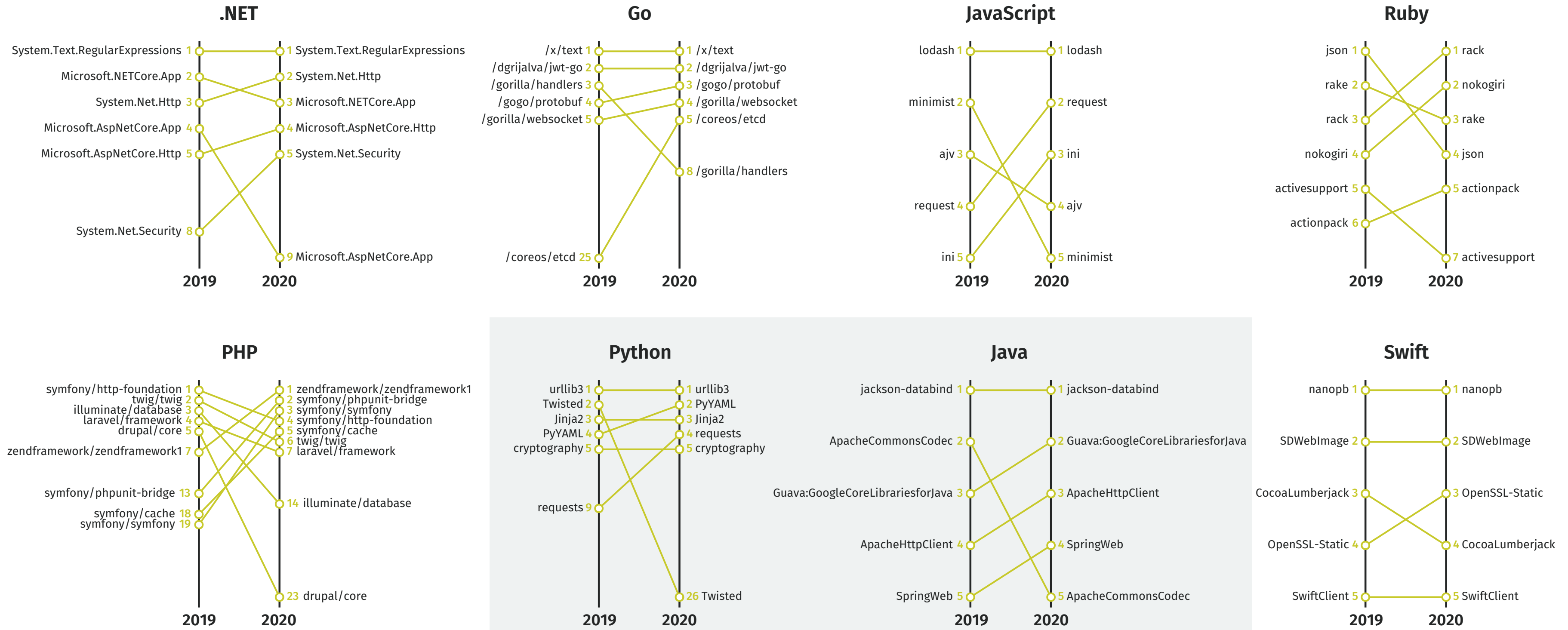


Figure 2 *Top vulnerable libraries from 2019 and 2020*

This year, we wanted to examine libraries that are both popular and have known vulnerabilities.

So, we created a similar figure, which focuses on libraries that had known vulnerabilities and were scanned in both 2019 and 2020. The results can be seen in Figure 2. What's interesting here is the reappearance of some names from Figure 1.

PYTHON

The fall of the Twisted library in Python may be attributable to the expanding capabilities of the built-in functionality in Python, with the built-in library asyncio receiving significant updates in 2016 and late 2018, and perhaps more importantly has only seen one CVE associated with it (CVE-2021-21330), in contrast to Twisted's seven over the course of its lifetime.

JAVA

Jackson-databind has both vulnerable and non-vulnerable versions but is so popular that it makes both lists.

https://t.me/learningnets

A point that we'd like to emphasize (though one that might seem obvious to most) is that what's hot and what's not can change within the span of a year. Probably more importantly, what's secure and what's not can change equally fast. Old libraries "age like milk" and so keeping up with an inventory of what's in your application is important.

https://t.me/learningnets

SECTION THREE

LIBRARY SELECTION

10 Library selection process
11 Library evaluation and problems
13 Most libraries are never updated
15 How long do they stick around before being updated?

We've looked at which libraries are selected most often, so now we take the next step and ask: How do developers choose libraries for their applications? After all, when borrowing someone else's code, there are a lot of things to consider:

- Does this do exactly what I want?
- Will this library introduce any vulnerabilities into my application?
- If a vulnerability in a library *is* discovered, how quickly will it be addressed by the library's developers?
- Does its license even permit me to use it in a commercial application?

We could go on and on (seems like we already have), but rather than keep asking questions, let's get to some answers. To help provide those answers, we surveyed Veracode users asking these questions, and more. We received nearly 1,800 responses to our short survey, and we were able to correlate survey responses to anonymous account data.[3] This allows us to correlate the responses in the survey with the actual development practices.

LET'S DIVE IN AND SEE WHAT WE CAN SEE.

[3] *See the appendix for some more details on our data.*

https://t.me/learningnets

Library selection process

We first asked, "Do you have a formal process for selecting third-party libraries?"

It is unsurprising that customers who care enough about software security to purchase scanning software, overwhelmingly, have a formal process for library evaluation, though the large fraction (29 percent) who are unsure is a bit concerning. This means that either they don't have a formal process or that they are unaware of the process they do have and may be simply ignoring it.

Developing, sharing, and following a unified policy can be difficult among large and disparate teams, likely leading to the uncertainty we see in Figure 3.

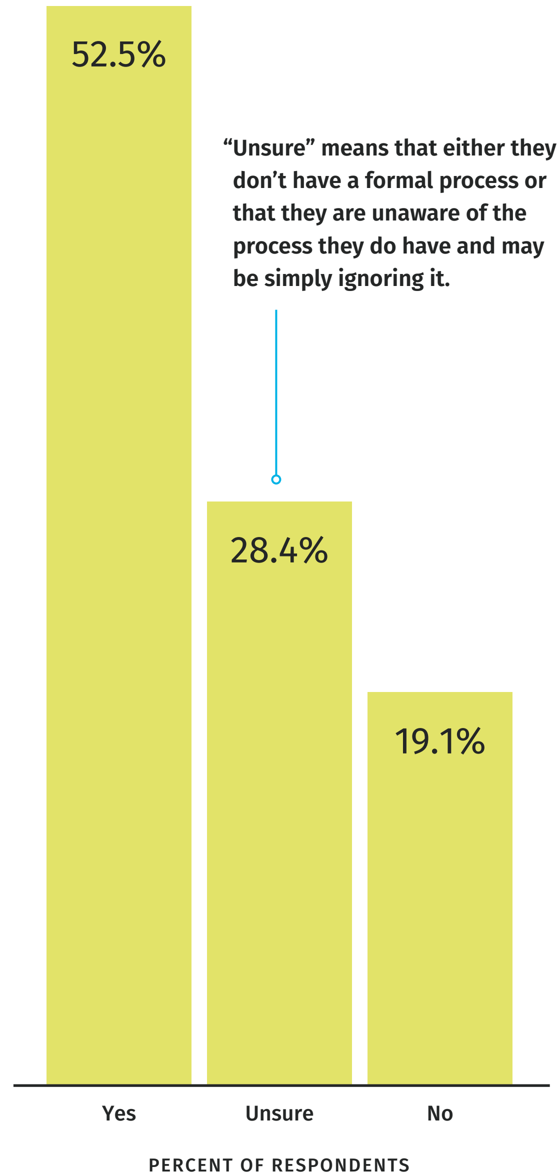


Figure 3

Number of organizations with a formal process for selecting libraries (n=1.5k)

WHAT ABOUT MY CONTAINERS?

Unless you've been living under a rock for the past few years, you've probably heard terms like "container" and "docker" and "Kubernetes" being tossed around the application development world.

Containerization has been a boon to developers much in the same way third-party libraries are. It allows them to not only package up their applications and libraries to run on some predefined server OS, but also allows them to bring the whole OS along with them.

This type of inclusion requires its own analysis, and we have some research coming down the pipeline looking into the unique challenges presented by containers.

https://t.me/learningnets

Library evaluation and problems

OK, so you have a process, but *what* is that process?

Specifically, we asked what developers look for when they are considering adding a new library. The results can be seen in Figure 4. Unsurprisingly, the leader here is functionality. After all, without the correct functionality, what's the point of including a big pile of code? Next in line is licensing, followed by security. It is not surprising that all of these are considered at least frequently by 80+ percent of respondents; all these things matter when selecting a library. What might be a better split here is whether they are *always* considered. This is an indication that a particular need is part of the selection process.

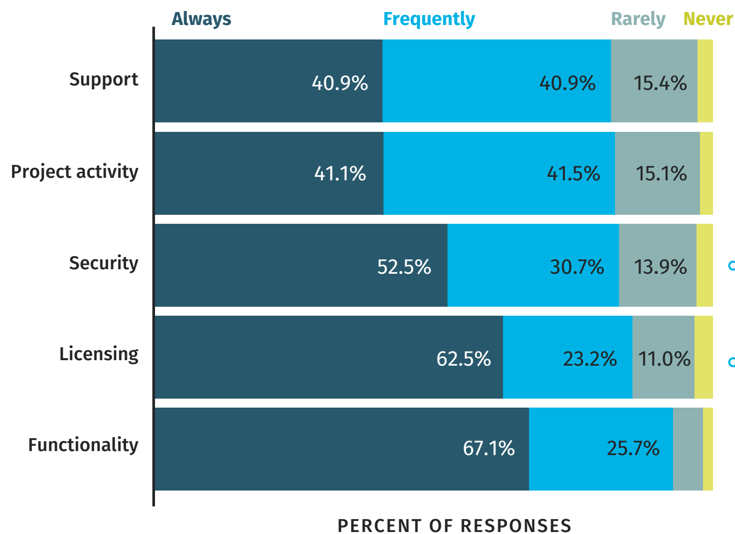


Figure 4 *Priorities when selecting libraries (n=724)*

[4] *We'd start with "functionality," but Veracode can't make a library do what you want it to do.*

To that end, we examined whether users who always considered one of the previous criteria have fewer issues in a particular area.

LICENSING

We start with licensing[4] in Figure 5, and we see that repositories are much more likely to have license issues on the latest scan of the default branch if survey respondents say they don't "always" consider the license when selecting libraries. This is a pretty stark divide. While known violations resulting in legal actions are rare, when they do occur, they can cost big money (up to $150k per instance). Ensuring that you are allowed to use a particular library and making that part of your evaluation process seems like a low-cost hedge against major future headaches.

PERCENT OF REPOSITORIES WITH LICENSE ISSUES

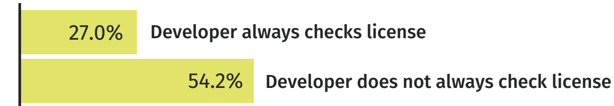


Figure 5 *Scanning for license issues resolves problems*

SECURITY

Licensing is certainly an issue, but what about the main event: Security? Figure 6 indicates that having a formal process reduces by a small amount the percentage of libraries in repositories that have vulnerabilities. Two issues present themselves in this result. First, as we saw last year, almost all repositories include libraries with some sort of vulnerability. The other is that the data is biased towards those who *do* think about security; they bought Veracode products after all. But even in the face of these two issues that would likely obscure any difference, we can still find one of about 3.5 percent.

PERCENT OF REPOSITORIES WITH VULNERABILITIES IN THIRD-PARTY LIBRARIES

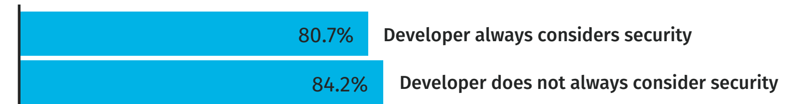


Figure 6 *Formal security process reduces vulnerabilities*

https://t.me/learningnets

Picking libraries is just the start; developers also need to maintain them. Security issues crop up, new functionality is added and old is deprecated, projects are abandoned. Taking care of these libraries while they are under development is a challenge. In this section, we examine how developers handle the changes to the libraries they use.

https://t.me/learningnets

Most libraries are *never* updated

One striking fact is that once developers pick a library/version, they tend to stick with it. Figure 7 shows that 65 percent of libraries appear in the first scan of the repository and are never updated, with an additional 14 percent added at some point during development and never updated to a new version.

I know what you are thinking, "Well, some of these repositories might yet be young, you might not have seen the full life of the application." Nope. Even if we restrict this to repositories that have relatively long lifespans and many scans, 73 percent of libraries are added and never updated (Figure 8). As with everything in application development, this depends on the language of course.

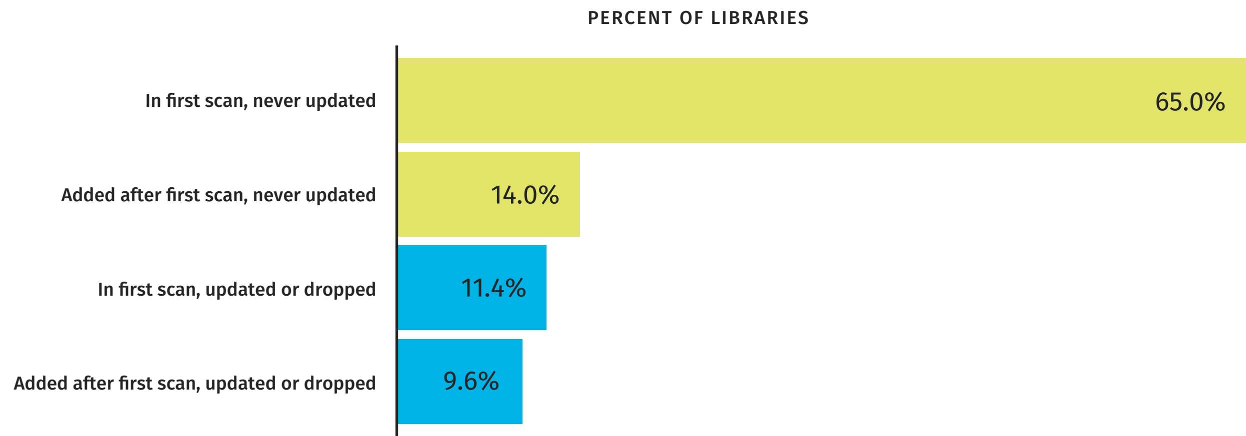


Figure 7 *How often developers update libraries*

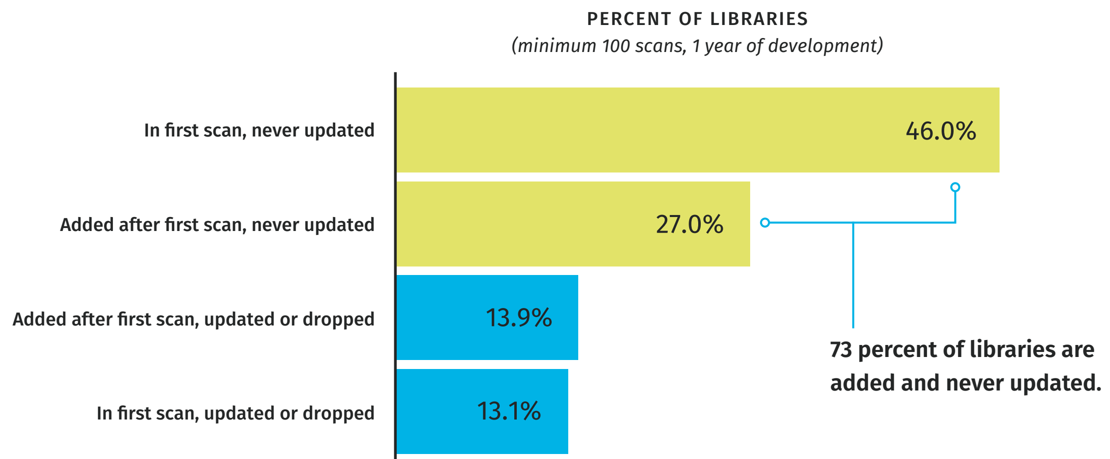


Figure 8 *How often developers update libraries over time*

https://t.me/learningnets

Restricting to repositories we have significant data on (100 scans over one year), we see that some languages get more attention than others. We examine the difference in Figure 9.

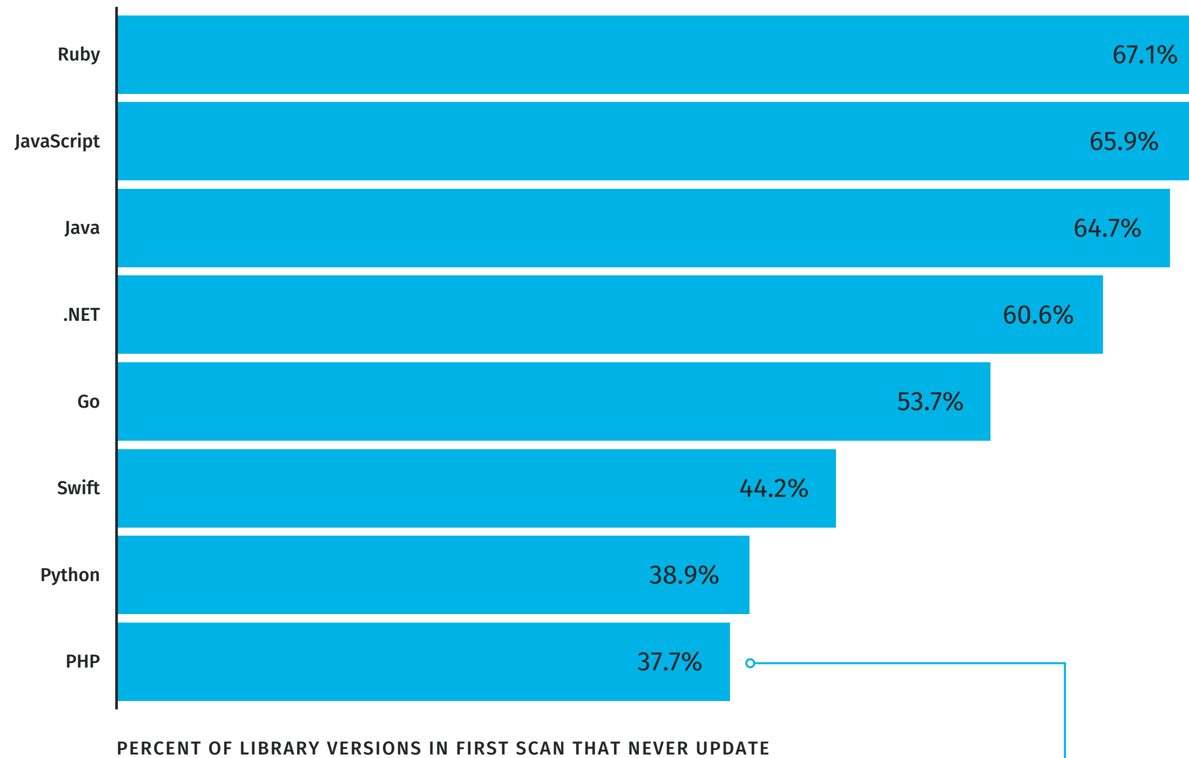


Figure 9 *Libraries never updated by language*

An interesting thing here is that PHP, usually the security black sheep among languages, has the lowest rate of "set it and forget it." While it shines here, sadly it won't last as we'll see later.

https://t.me/learningnets

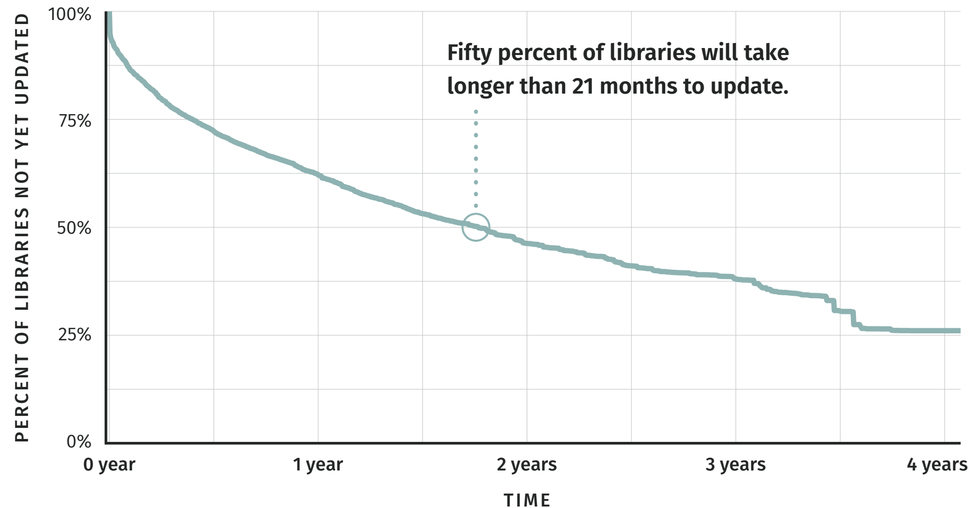


Figure 10 *Time to update libraries*

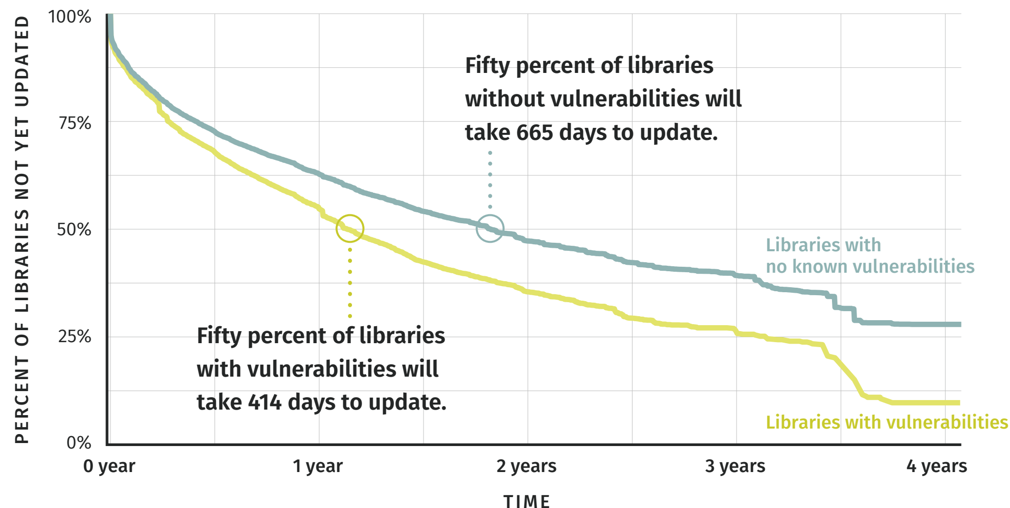


Figure 11 *Time to update vulnerable libraries*

How long do they stick around before being updated?

Many libraries have never seen an update, even after one year, but how long does it take to update those that actually do get updated?

We use a fancy statistical technique called survival analysis.[5] Survival analysis is a technique developed mainly to understand the survival time of patients facing various types of diseases and inferring the effects of treatment.[6] In short, it works like this, we look at the lifetime of those we know are updated and use that to estimate how long the ones that haven't been updated yet will stick around. The results can be seen in Figure 10.

Consistent with the "most libraries have never been updated" stat, survival analysis estimates that libraries stick in applications for a *very* long time. Fifty percent will take longer than 21 months to update, with an estimated 25 percent not being updated after as long as four years (the time horizon of our data).

But we are here to talk about security, so let's not just think about how long it takes developers to update to the next version, but how long it takes them to fix *vulnerable* libraries, and good news, vulnerable libraries are updated faster!

Figure 11 shows that it takes about 665 days for 50 percent of libraries without vulnerabilities to be updated, but only 414 for those with vulnerabilities. Programming note, this doesn't mean developers are taking more than a year to update or fix once alerted of a vulnerability. Rather, this includes the time when the library is used but isn't known to have a flaw. In the next section, we examine the reaction time of developers once they know there is a flaw.

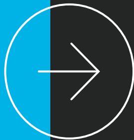
[5] In particular we are using a Kaplan-Meier estimate to understand the time to update libraries.
[6] True SOSS fans will remember us using this kind of analysis going back years.

https://t.me/learningnets

SECTION FOUR

FIXING VULNERABILITIES

19 Severity
20 Dependency type
21 Vulnerability type
22 Language
23 Developer resources



https://t.me/learningnets

The previous sections asked how long it takes to update vulnerable libraries (whether the vulnerability is known or not), but this excludes the time the vulnerability was unknown or the time it was known, but the developers weren't notified. Once a developer sees the result of the scan on their repository, how fast do they react?

The answer in Figure 12 is pretty darn fast. This chart is going to act as our Rosetta stone for the next few charts.

As we do more 'time to update vulnerable libraries' curve comparisons, things can get awfully cluttered. So we've simplified the curve to the right into a segment.

PERCENT OF VULNS

The "50% of vulns" acts as a measure of 'typical', while 25 percent and 75 percent give us a good sense of how quickly the curve descends. SOSS fans will remember these 'interval' charts appearing in SOSS Volume 9.

X-AXIS

One last note, time on the horizontal axis is on a 'log scale', that means each step is an order of magnitude increase rather than a fixed time period. If that's too much, don't worry, we'll directly label the points so you can make your own comparisons.

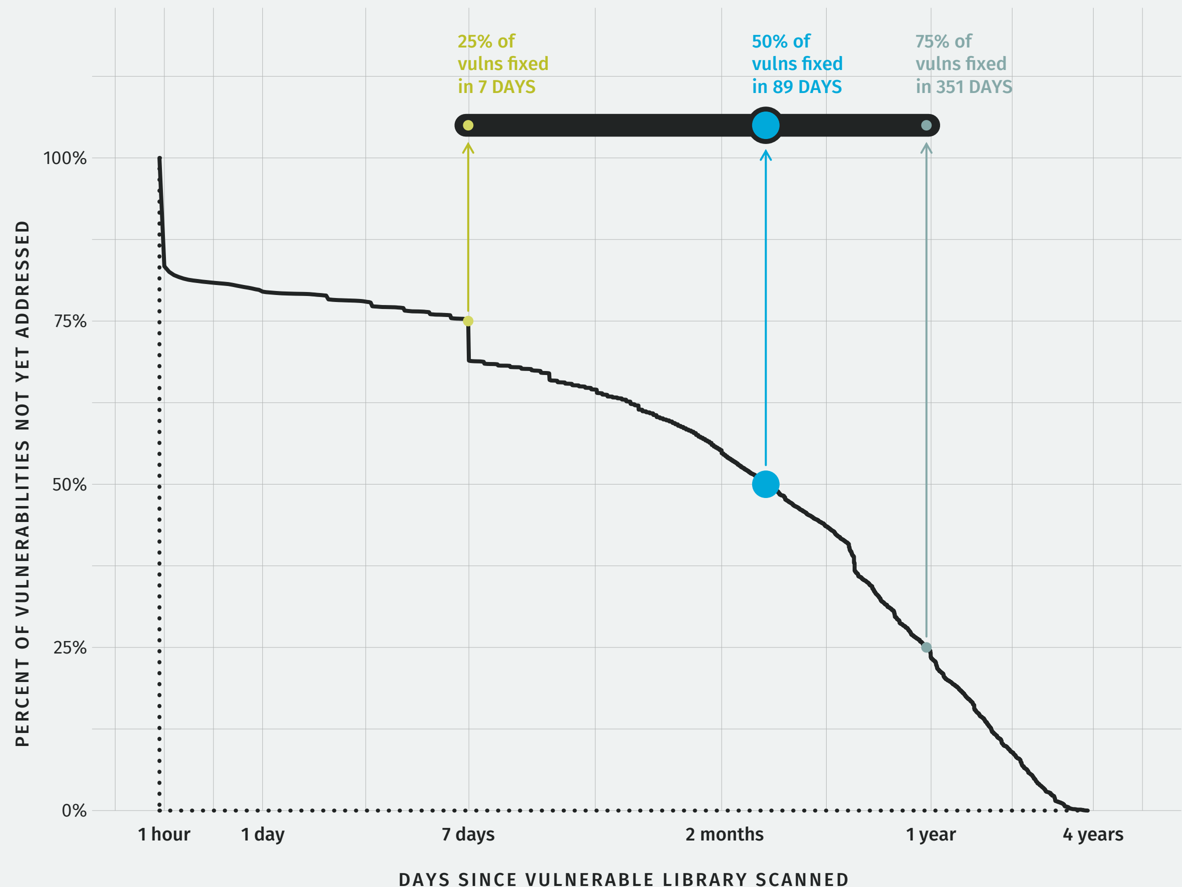


Figure 12 *Time to fix vulnerable libraries once alerted to the issue*

https://t.me/learningnets

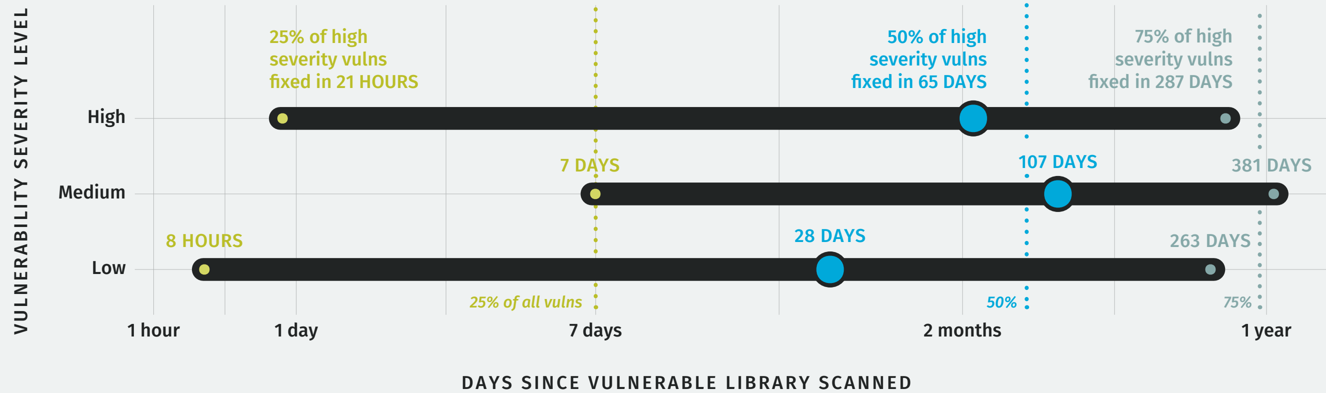


Figure 13 *Library update speed based on flaw severity*

In fact, nearly 17 percent of vulnerable libraries are fixed within an hour of the scan that alerted the developer to the vulnerability; 25 percent are fixed within seven days. The bump seen in Figure 13 indicates that many developers probably scan weekly, see the vulnerabilities, and update. After that, 50 percent of vulns are fixed within three months, and 75 percent within a year.

Some vulnerabilities linger, and we'll look at what causes that lingering shortly. The kernel of truth in Figure 13 is that once developers are made aware of flaws they can (and do!) take action quickly. Having the right information makes applications more secure, faster.

TIME IT TAKES TO FIX VULNERABLE LIBRARIES

Within one hour
17%

Within three months
50%

Within one week
25%

Within one year
75%

https://t.me/learningnets

Severity

So now we know that most vulnerable libraries get updated quickly, but are developers prioritizing the dangerous ones first?

Veracode tracks vulnerability severity, partially based on CVSS score, so we can see whether high-severity vulnerabilities get addressed first. Interestingly, no clear trend emerges in Figure 13. Low-severity vulns are fixed the fastest, and high-severity vulnerabilities are fixed *slightly* faster than the population at large. As a bit of foreshadowing, allow us to speculate that other factors are driving the replacement of libraries.

It's possible that most developers are unconcerned with severity, but luckily we had the foresight to ask developers if this was a factor, and, lo and behold, some, but not all, respondents *do* care about severity. If we look at survey respondents and see if they consider severity in Figure 14, we find that those developers fix low- and medium-severity issues more slowly, and high-severity issues much more quickly, exactly as you'd expect.

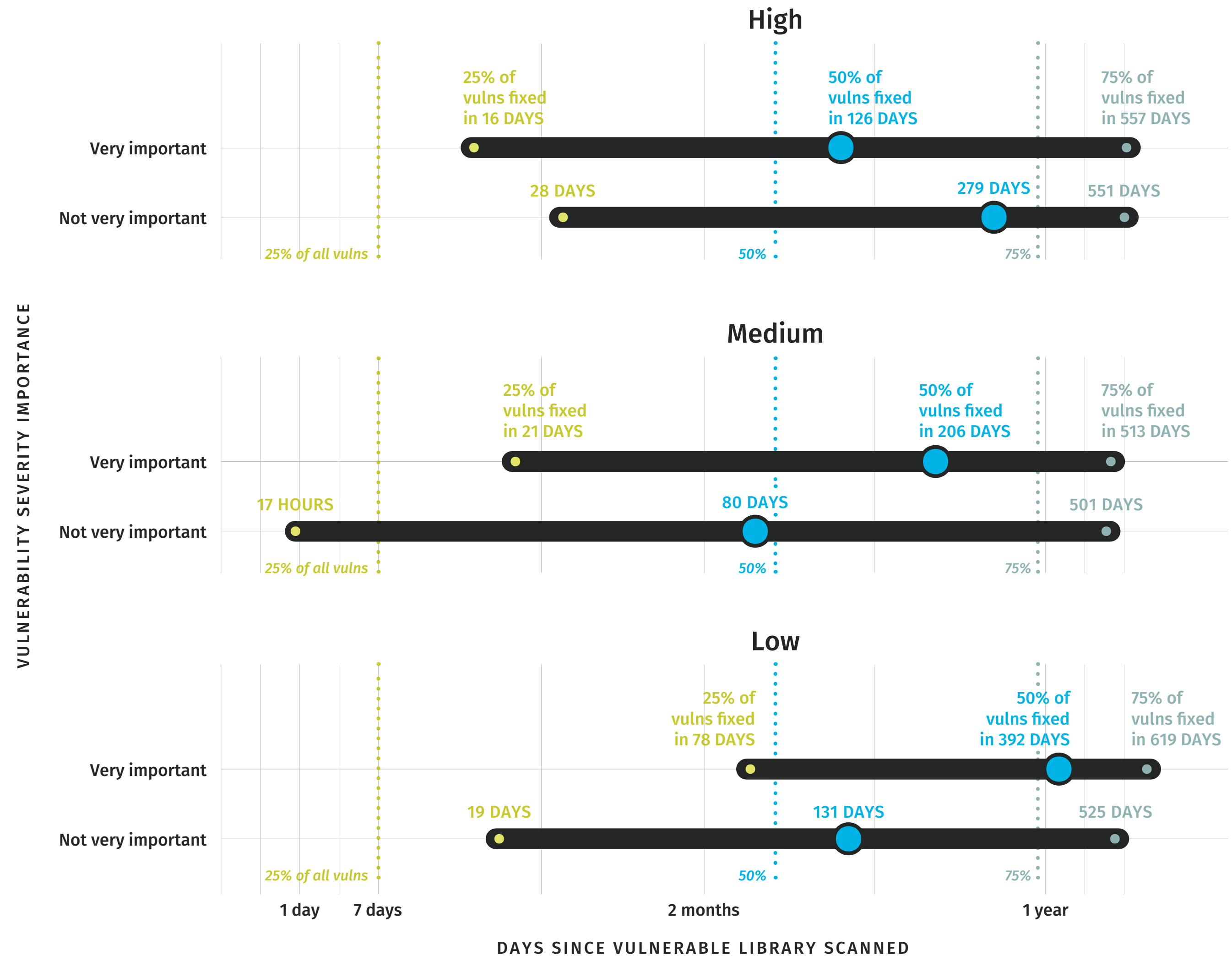


Figure 14 *Effect of prioritizing severity of security issues on update time*

https://t.me/learningnets

Dependency type

If it's not severity affecting update time, it may be something else like exactly how intertwined a particular library is with your project.

So, we next examined how different dependency types affect the speed of updating to non-vulnerable versions. What we see again easily fits our intuition in Figure 15. Direct dependencies are the easiest (fastest) to fix. Things get trickier with transitive dependencies; it may be that a fix will break some functionality in the direct library, meaning a slower and more difficult fix process.

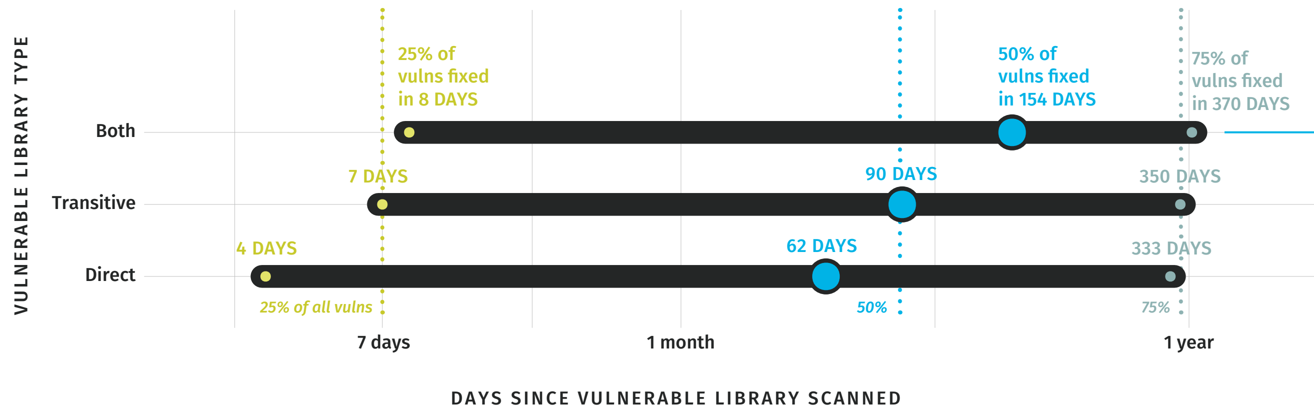


Figure 15 *Effect of library dependency type on update time*

When a library is both a direct and a transitive dependency, things get complicated with fixes taking nearly 2.5 times longer.

https://t.me/learningnets

Vulnerability type

Another aspect of complexity may be exactly what the nature of the vulnerability is.

We can imagine things that affect the fundamental functionality of a library might take the library developers a while to address and the patch might alter the functionality of the library. The particular type of vulnerability (here by CWE) is at least a partial description of this complexity, and Figure 16 looks at the top 10 most commonly seen vulnerabilities.

Vulnerabilities that we expect to be complex, such as "Arbitrary Code Execution," take a significantly longer timespan to fix than a typical vuln (187 days as opposed to the 89 days across all vulnerabilities). In contrast, things like Prototype Pollution should be relatively easy for library developers to address, i.e., one additional line that checks to make sure user provided objects don't try to modify __proto__ attributes. So why would we expect to see a difference in fix time for those just using the libraries? If a flaw is complex for library developers to fix, it may require fundamental changes to the way the library operates, making integrating those changes into downstream applications harder.

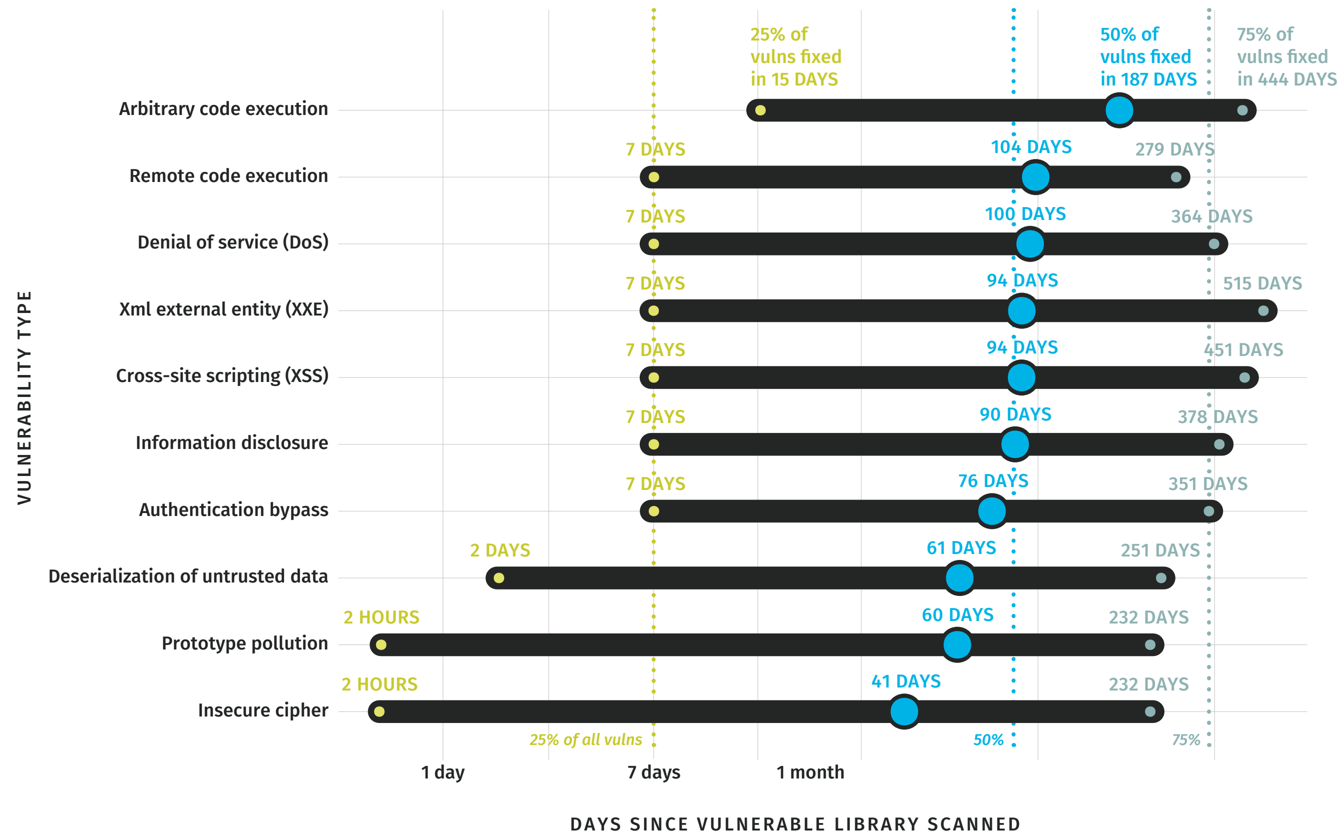


Figure 16

Effect of vulnerability type on library update time

https://t.me/learningnets

Language

Once again, we feel obligated to drive home the point that nearly everything depends on language, and fix times are of course no exception. There are some remarkable results in Figure 17.

ORDERING

First, the ordering (here by the time to resolve 50 percent of library vulnerabilities) is unusual from how "secure" different libraries in different languages were last year. For example, PHP had a high percentage of libraries with vulnerabilities, a high density of those flaws, and a high percentage of flaws with Proof of Concept exploits publicly available.[7] But here we see that half of vulnerable libraries in PHP applications are fixed in a little over two months, the third-fastest among languages. A heaping dose of surprise at this result is due to the fact that last year we saw PHP performing dead last when examining flaw density in libraries and the number of flaws introduced into applications by PHP libraries.

SPEED

Two frankly bonkers results here are the speed of Python and JavaScript. Both manage to fix 25 percent of vulnerabilities in less than five hours, with Python applications addressing 50 percent of flaws the same hour they are reported. The tails here are long, though. For most languages, flaws will stick around for years, and with some languages (.NET, Go, and Ruby), a not insignificant number of flaws (17 percent, 10 percent, and 6 percent respectively) are never going to be fixed within the time horizon of our data.[8]

Figure 17 *Time to update insecure libraries by language*

[7] *State of Software Security Open Source Edition 2020.*

[8] *"Forever vulnerabilities."*

https://t.me/learningnets

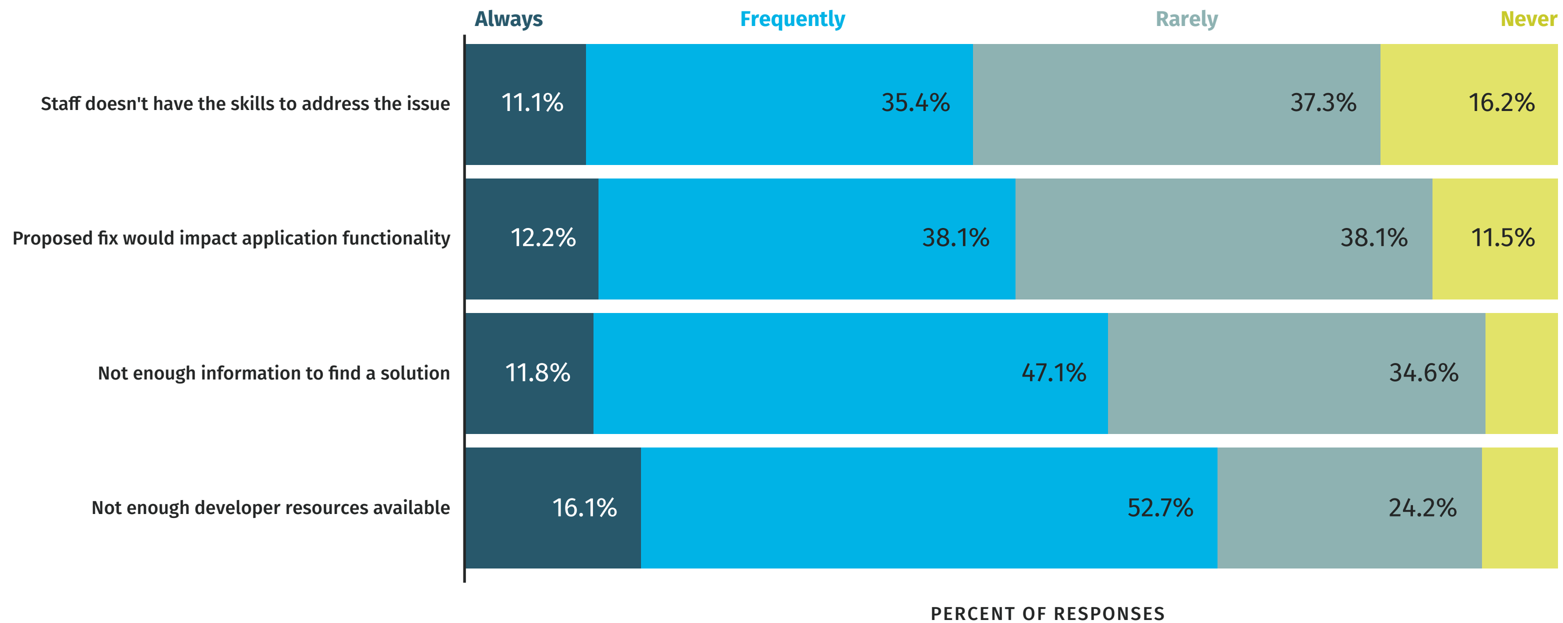


Figure 18 *Hindrances to addressing vulnerable open source libraries*

Developer resources

Aside from the nature of vulnerabilities (what language they are written in, their type, and how they are included in an application), there may be exogenous factors slowing developers down.

As part of our survey, we asked respondents how each of the following factors affected their ability to address vulnerabilities in third-party software (Figure 18).

The good news is that the majority of respondents are rarely (or less) lacking in skills to fix things, but often a lack of information or a time crunch can lead to roadblocks.

https://t.me/learningnets

What's remarkable is we see a large split in fix times based on those who answer "Often" (Always or Frequently) vs "Rarely" (Rarely or Never) in Figure 19.

While it is unsurprising that developers who say they struggle do in fact struggle, it's the scale of that struggle that is staggering. When developers frequently don't have the resources to fix vulnerabilities, it can take nearly 13.7 times longer to fix half of them.

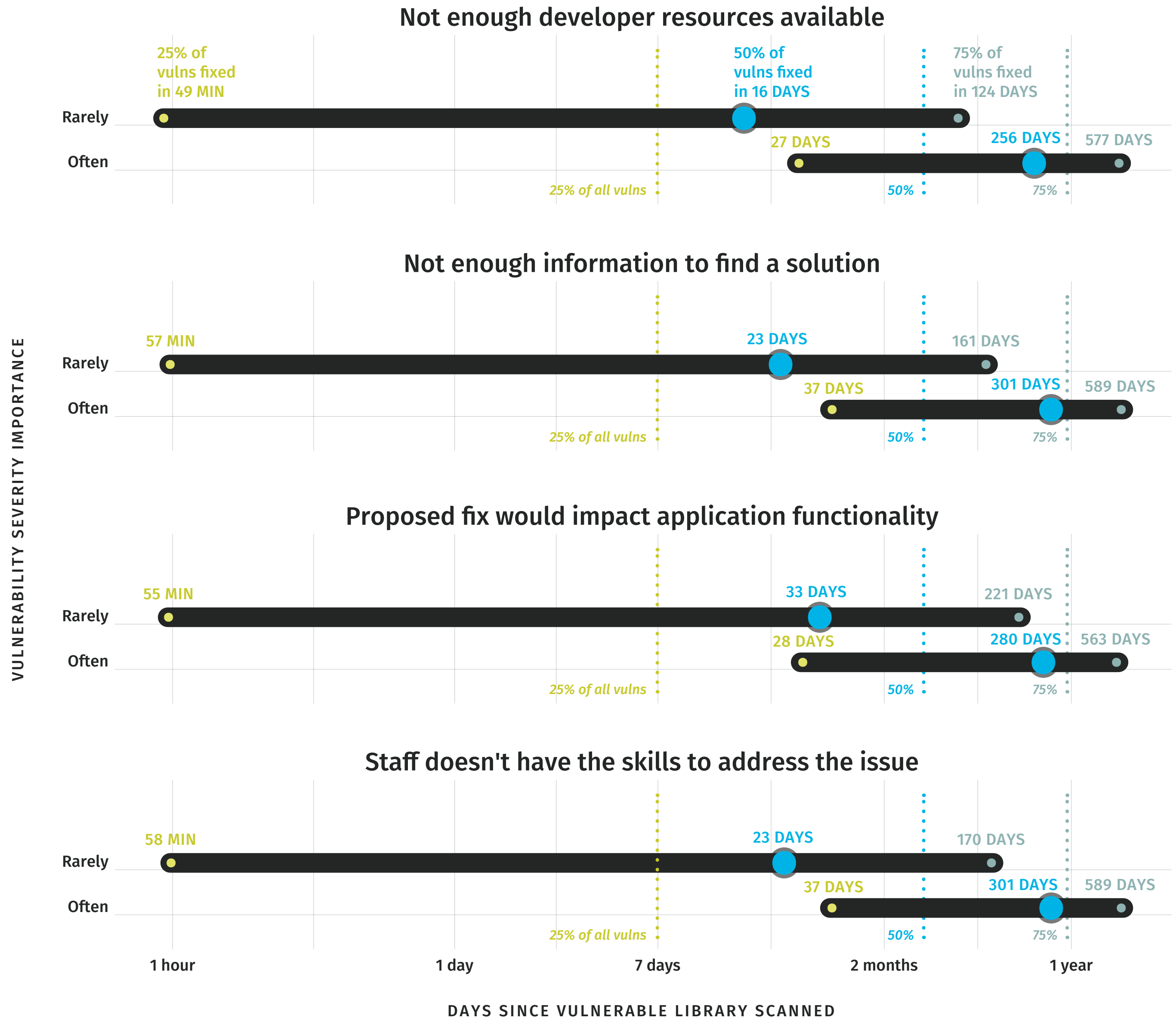


Figure 19 *How different hindrances affect time to fix vulnerable libraries*

https://t.me/learningnets

SECTION FIVE

SUGGESTED UPDATES

26 Most updates are still small

30 Update chains

One of the most encouraging findings from last year's report is that "most fixes are small." By that, we mean most findings require updating a library to a new version, which only differs by a minor, patch, or revision number. If semantic versioning is being followed, the implication is that most vulnerable libraries can be updated without a major impact on functionality.

We dive a bit deeper into what we mean by most updates being small and what happens when updates induce their own problems in this section.

LET'S SEE WHAT WE CAN SEE.

https://t.me/learningnets

Most updates are *still* small

This year, we are working with a slightly expanded dataset, and a whole new year of vulnerabilities and library development has shifted our distribution a bit. We've got good news and bad news. The good news is there are fewer vulnerable libraries that don't have an update available with a fix (26.2 percent down to 8.4 percent). Unfortunately, most of that change goes straight into larger updates, which now make up 31 percent of all updates (Figure 20).

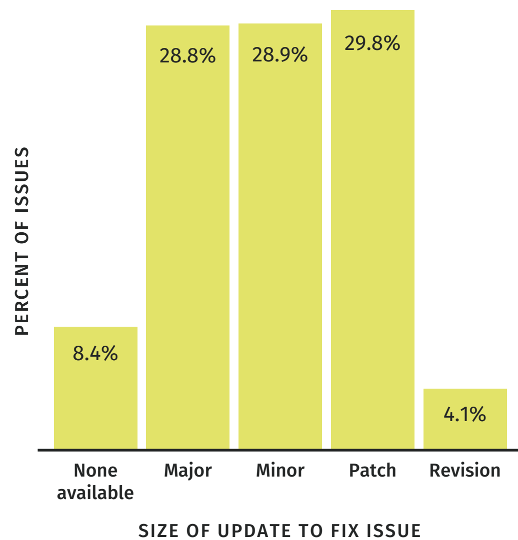


Figure 20 *How complicated are library security fixes?*

When we ended on this fact last year, we didn't slice and dice the results. This year, we can't help but break out the metaphorical knives and view this fact through the lens of various factors we know make a difference in other parts of the DevSecOp world.

BUT WHAT ABOUT WHEN THERE AREN'T UPDATES?

We've dug in hard on what updates look like for library fixes, and with good reason, as we saw less than 10 percent of vulnerabilities in third-party libraries don't currently have updates that allow them to be fixed. So what do developers do when faced with this minority? In our survey, we asked. And Figure S1 has the answers.

Most are going to look elsewhere for the functionality (70 percent responding "Always" or "Frequently"), or just do it themselves. Unfortunately, contributing a fix to the library itself is somewhat of a rare occurrence, but it does happen, with nearly 10 percent saying they "Always" do. A deeper dive into those unfortunate 8.4 percent of flaws with no update available and how they might be addressed is certainly rich ground for future work.

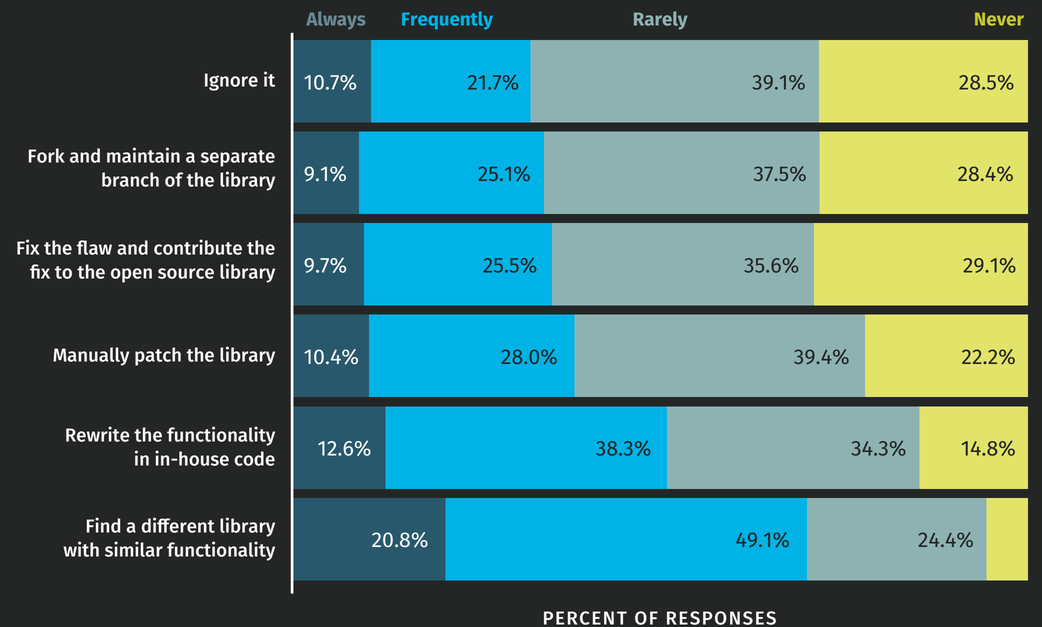


Figure S1 *Actions taken when no update exists for a vulnerable library (n=279)*

https://t.me/learningnets

LANGUAGE

First and foremost is language, and in a return to the familiar once again, Figure 21 has bad news for PHP.

PHP

While the vast majority of vulns can be fixed with an update in PHP, more than 60 percent of them require a major update. We can't help but commend PHP developers for being relatively fast fixing their libraries in spite of the fact that it usually requires a major version bump.

JAVA

Another good news/bad news situation is with Java, which has the highest percentage of flaws that can be fixed with a minor update or less, but the second highest number that don't have any update currently available.

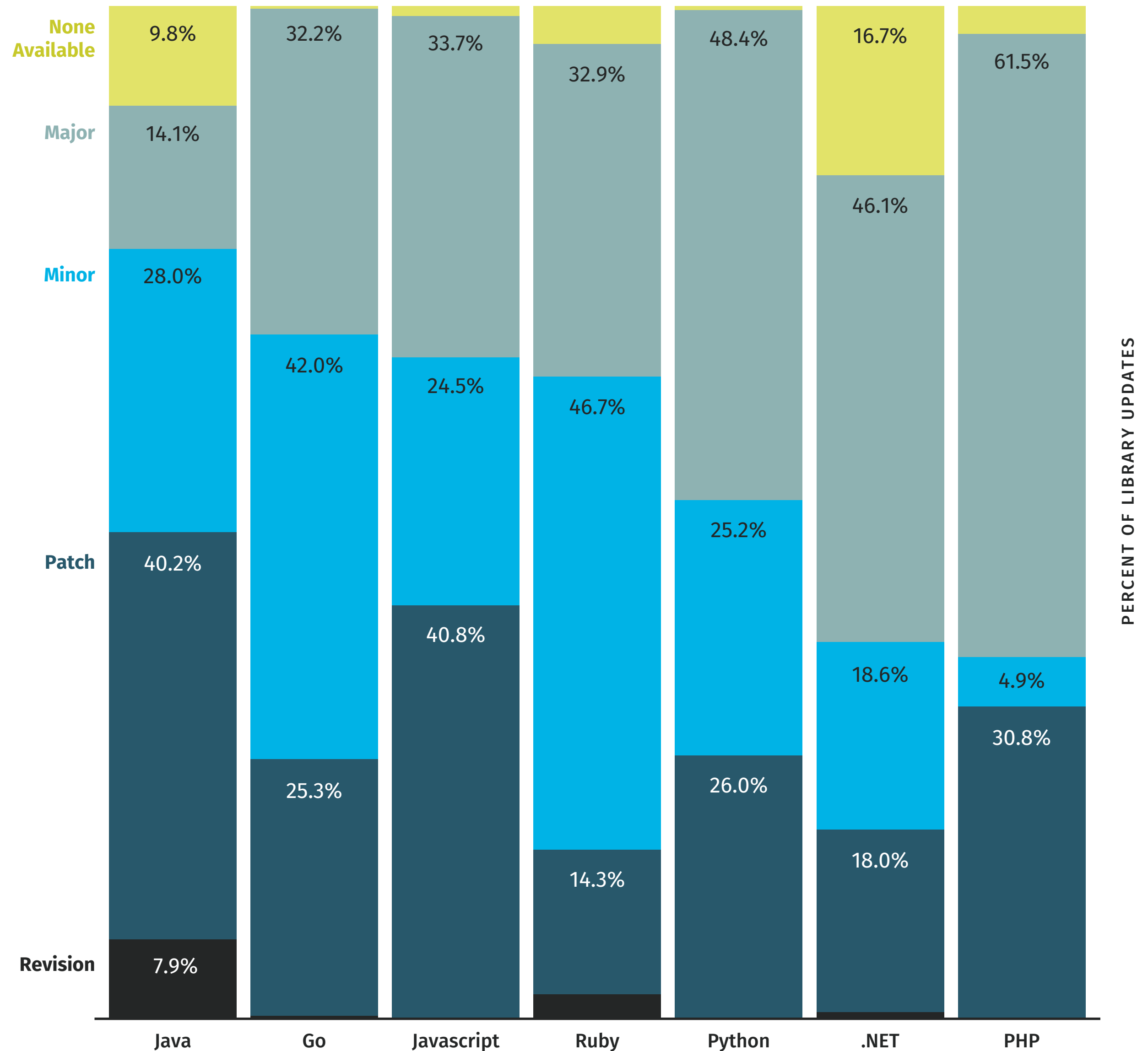


Figure 21 *Scope of library security fix by language*

https://t.me/learningnets

DEPENDENCY TYPE AND SEVERITY

For completeness, we also present the breakdown based on severity and dependency type in Figure 22.

There is some variation here, but we have no (statistical) reason to believe that the severity is actually altering the distribution in significant ways. Are high-severity vulns slightly more likely to have small updates? Sure, and that's probably a good thing, but it's not a striking difference. In the same way, we don't see a huge difference due to the way the library is introduced into an application.

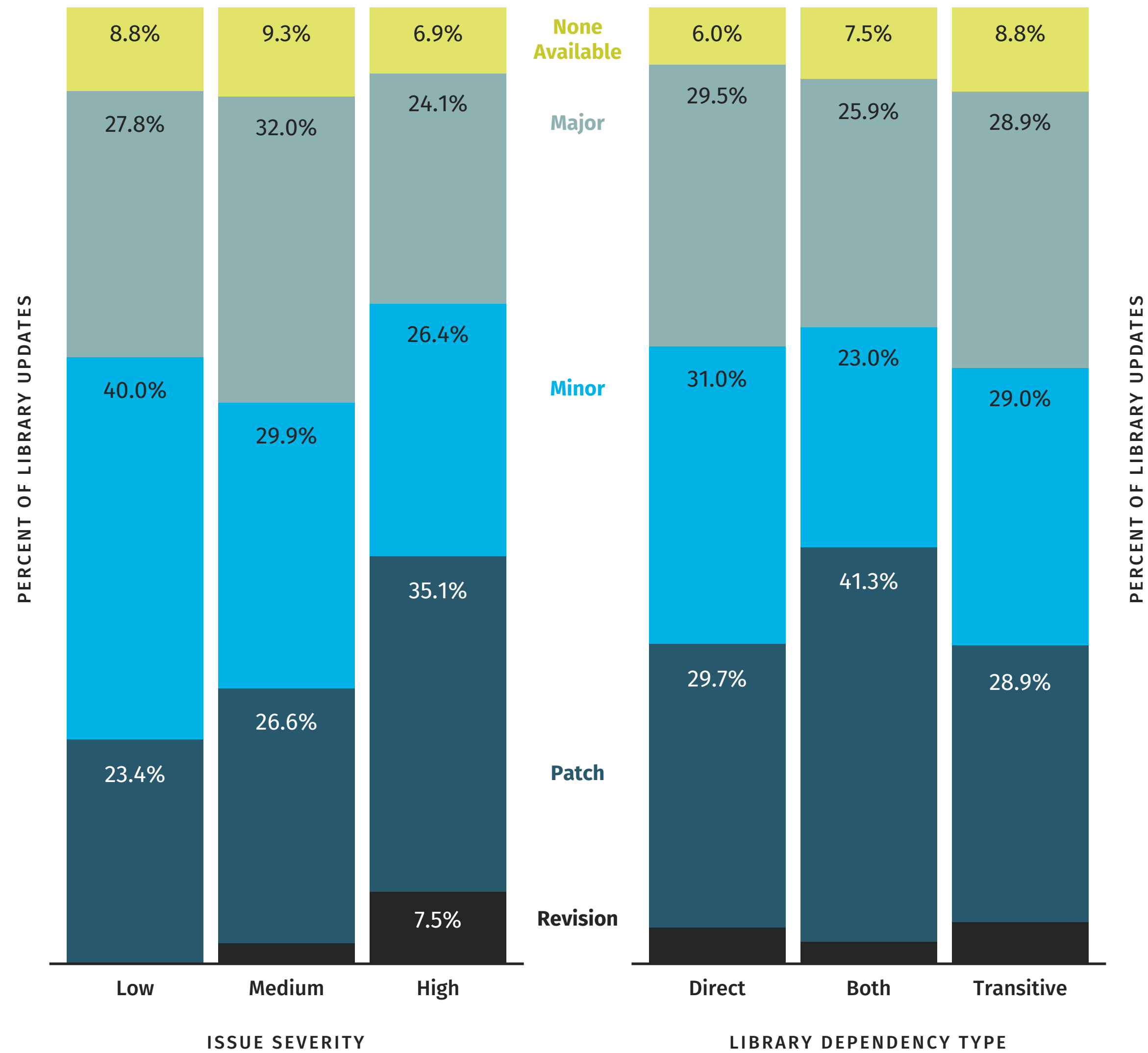


Figure 22 *Size of update required by dependency type and flaw severity*

https://t.me/learningnets

TYPE OF FLAW

Where we do see significant variation is in Figure 23. Things that often require major updates are fundamental to a library's functioning.

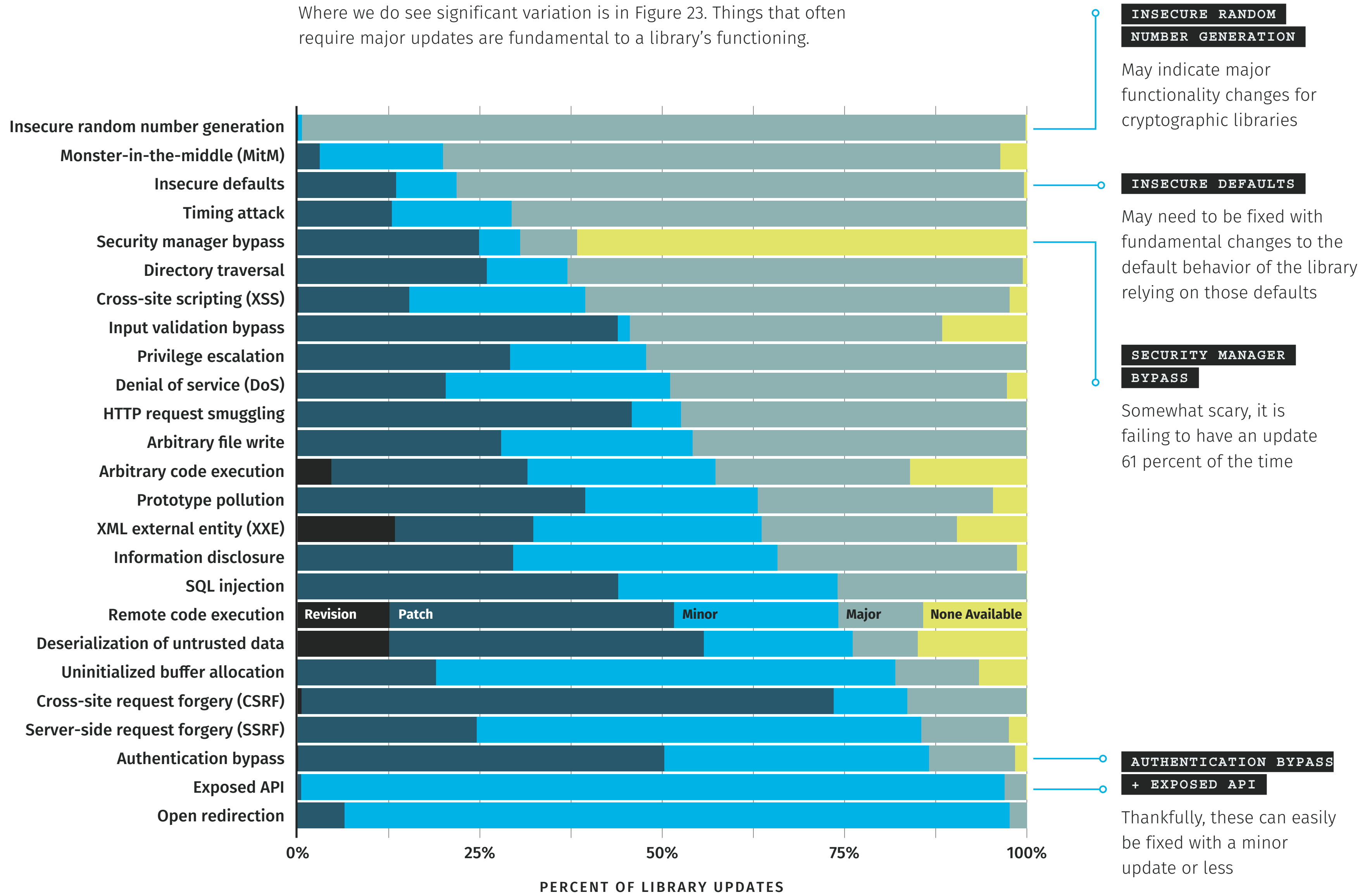


Figure 23 *Size of library update required by vulnerability type*

https://t.me/learningnets

Update chains

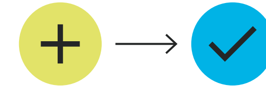
Any experienced developer will recognize that updating a library to a new version may not be the end of things.

Indeed, updating a library might simply beget new vulnerabilities, requiring more updates, which beget new vulnerabilities which...you get the idea. We examine these chain updates here. First, we need to think about what the possibilities are for various types of update chains.

01.

ONE STEP TO UNFLAWED VERSION

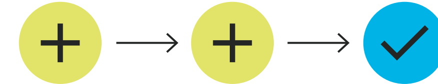
A single update fixes all our problems.



02.

MULTIPLE STEPS TO UNFLAWED VERSION

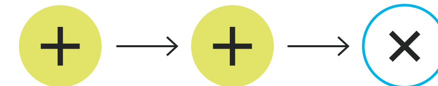
One update is not enough, but after enough steps, we get to a clean version.



03.

MULTIPLE STEPS TO FLAWED VERSION

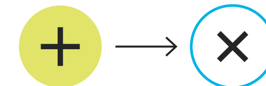
We do multiple updates, only to arrive at a flawed version with no further available updates.



04.

NO UPDATE AVAILABLE

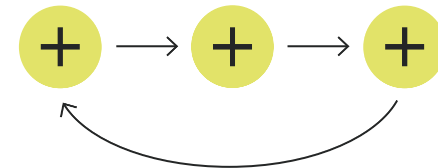
There might not be any update available to start with.



05.

CIRCULAR UPDATE

There might be a dreaded situation where the suggested updates are actually a downgrade to a version we've already updated to. These types of circular updates are likely the most pernicious to address.



https://t.me/learningnets

The relative breakdown of these possibilities is presented below in Figure 24, and the results are heartening. The slim majority of updates are a single step to a clean version, and most updates (more than 86 percent) end in a library with no known flaws. Thankfully, none of those scary updates appear in our data.

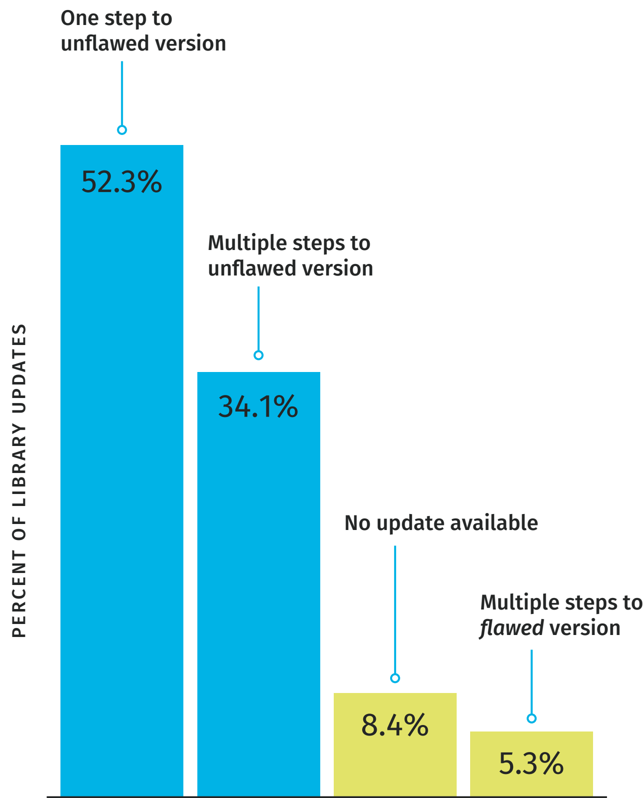


Figure 24 *Steps needed to update a vulnerable library*

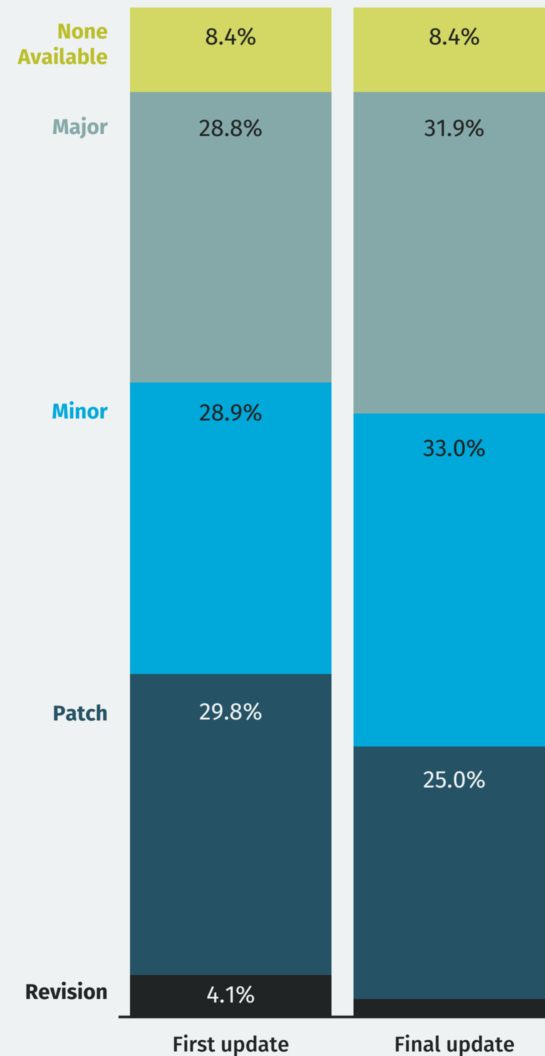


Figure 25 *Size of update required for vulnerable libraries based on step in chain*

But how do these chains affect our result that most updates are small?

It doesn't matter if the first update is small if the last update is large. Things shift a little towards larger updates in Figure 25, but it is not substantial. So the rabbit hole might twist and turn, but it generally doesn't take you too far from where you started.

https://t.me/learningnets

But exactly how deep is that rabbit hole?

It's one thing to say that the end result isn't a large update, but going through the pain of potentially dozens of updates might break the spirit of even the most strident developer.

FIGURE 26 SHOWS US

01.

Most update chains (when they do exist) are short.

02.

Long update chains themselves are surprisingly not correlated with particularly large updates, and in fact we see a rather random amount of variation, with the caveat that we have relatively small sample sizes for the medium length (three to five step) chains.

03.

Finally, and perhaps most surprisingly, long chains are less likely to dead end. In fact, chains longer than two steps in our data are guaranteed to end in a clean fix, so all that effort wading through dependency hell will eventually get you a less vulnerable application.

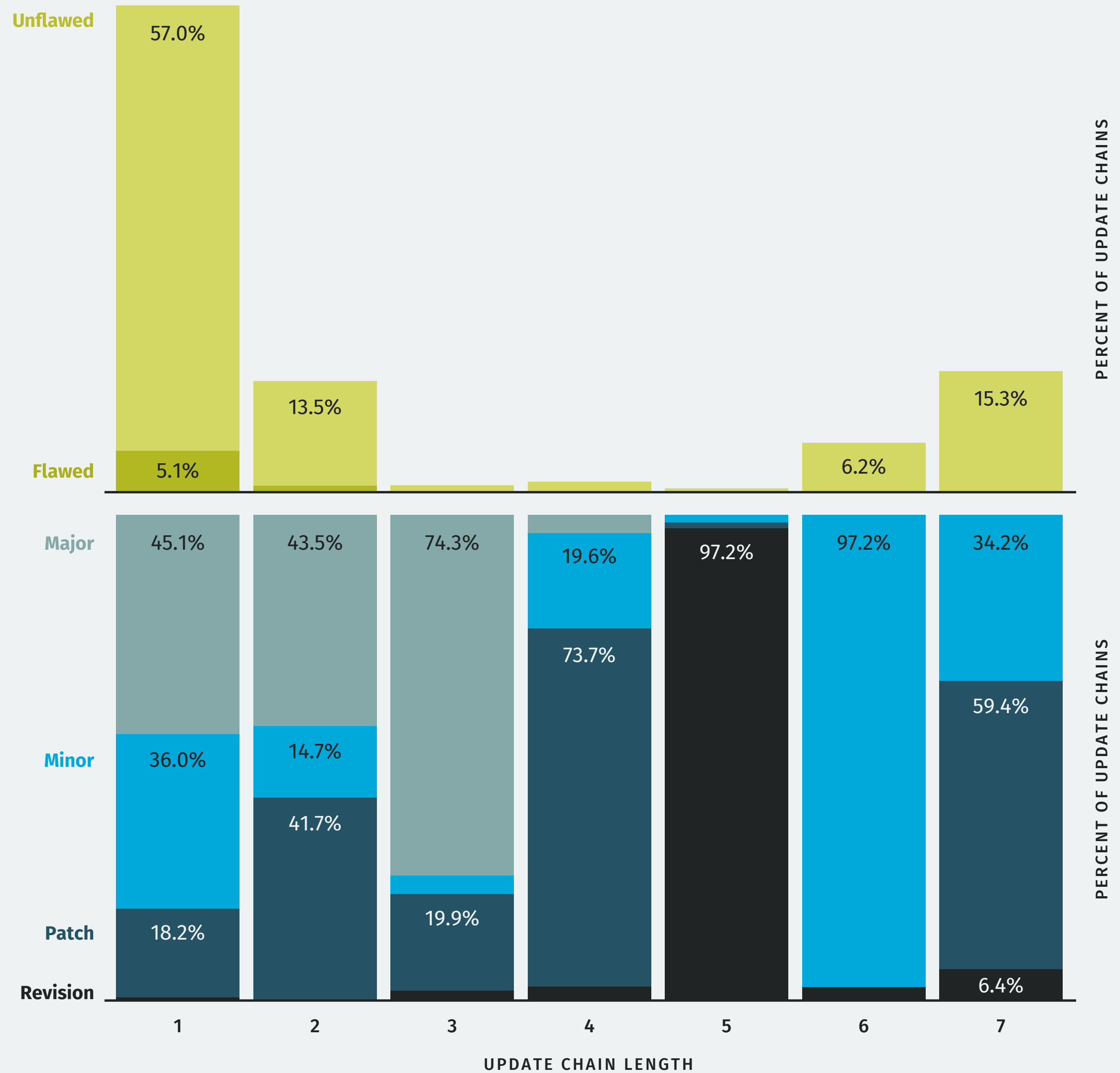
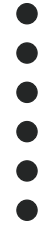


Figure 26 *Number of steps required to reach a secure library*

https://t.me/learningnets



SECTION SIX

CONCLUSION

Open source libraries are constantly evolving and changing.

Despite this dynamism, a large amount of library selection is "set it and forget it," with developers finding the functionality they need and never changing it. What was a functional library with no flaws two years ago may expose an application today.

So how do we face the challenge of this changing landscape? The results in this report suggest that when developers are given the information they need, they can act quickly to resolve issues. It helps that most fixes are no more taxing than a minor software update, something not likely to break the inner workings of even the most complex application.

TO LEARN MORE ABOUT SOFTWARE SECURITY, CONTACT US.

https://t.me/learningnets

Appendix: Methodology

This research draws on Veracode Software Composition Analysis to catalogue the use of third-party software. Customer repositories are examined for third-party library information and dependencies, generally collected through the application's build system. This includes nearly 13 million scans of more than 86,000 repositories, containing more than 301,000 unique libraries. Data on scans between July 2016 and February 2021 were examined. Libraries are checked against a database of known flaws, which includes the national vulnerabilities database. Suggested updates for vulnerable libraries are drawn from information about the particular vulnerability and the smallest update which addresses the flaw is considered.

This year anonymized account data was combined with an anonymized survey of Veracode customers through the Veracode platform. The survey received 1,744 responses from customers of a variety of our solutions. A fraction of survey respondents failed to complete the survey at each stage, each of the survey results is presented with the number of complete responses received.

Number of scans

13 Million

Number of repositories

86,000+

Number of unique libraries

301,000+

Survey responses

1,744

https://t.me/learningnets

VERACODE

Veracode is the leading AppSec partner for creating secure software, reducing the risk of security breach and increasing security and development teams' productivity. As a result, companies using Veracode can move their business, and the world, forward. With its combination of automation, integrations, process, and speed, Veracode helps companies get accurate and reliable results to focus their efforts on fixing, not just finding, potential vulnerabilities. Veracode serves more than 2,500 customers worldwide across a wide range of industries. The Veracode cloud platform has assessed more than 14 trillion lines of code and helped companies fix more than 46 million security flaws.

www.veracode.com

Veracode Blog

Twitter

Copyright © 2021 Veracode, Inc. All rights reserved.
All other brand names, product names, or trademarks belong to their respective holders.

https://t.me/lear