VOLUME 12

State of Software Security

The Progress We've All Made

VERACODE

01

Contents

SECTION 01

Executive Summary

The world is becoming more connected than ever before ... Connectivity makes our lives easier, but it also increases risk. One security flaw can have a domino effect, leaving software vulnerable all across the globe.

But it's not just increased connectivity that's shaping the security landscape — it's the hypercompetitiveness and the need to constantly innovate. To move faster, many development teams have turned to native cloud technologies, microservices architectures, and open-source code to accelerate and scale their efforts. Additionally, development teams have adopted agile methodologies and are automating as many steps in the development process as possible.

While this evolution increases the speed of the software development lifecycle, it also introduces new complexities and risks.

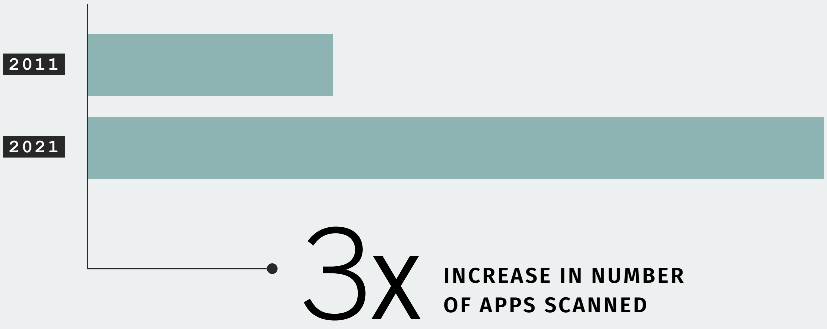
For our 12th State of Software Security report, we'll explore these trends with the help of the Cyentia Institute to assess how the software security landscape is continuing to evolve. Our goal is to help you make informed decisions about your software security program so that you can minimize your risk and meet cybersecurity regulations like those outlined in the White House Executive Order on Improving the Nation's Cybersecurity issued on May 12, 2021.

The State of Software Security at a Glance

Similar to last year, we looked at the entire history of active applications, not just the activity associated with the application over one year. By doing so, we can view the full life cycle of applications, which results in more accurate metrics and observations. Aside from looking at the past, we also imagined the future by considering practices — such as Veracode Security Labs training — that might help improve application security.

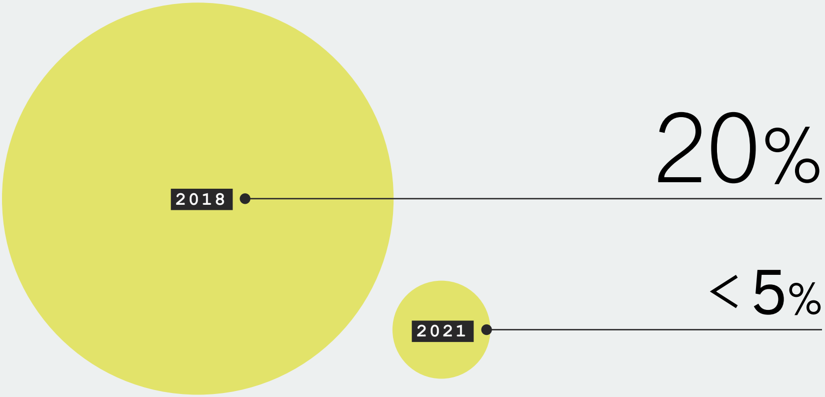
The Number of Apps Scanned Has Tripled

Organizations are scanning, on average, more than 17 new applications per quarter. This number is more than triple the number of apps scanned per quarter a decade ago.



Microservices

In 2018, roughly 20 percent of applications incorporated multiple languages. This year, less than 5 percent of apps used multiple languages, suggesting a pivot to smaller, one-language applications or microservices.



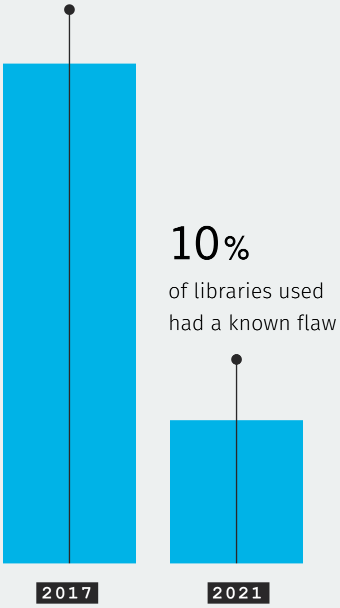
JavaScript, Python, and .NET have seen declines in app sizes, indicating a trend toward more microservices.

Flaw Prevalence

Third-party libraries have fewer flaws.

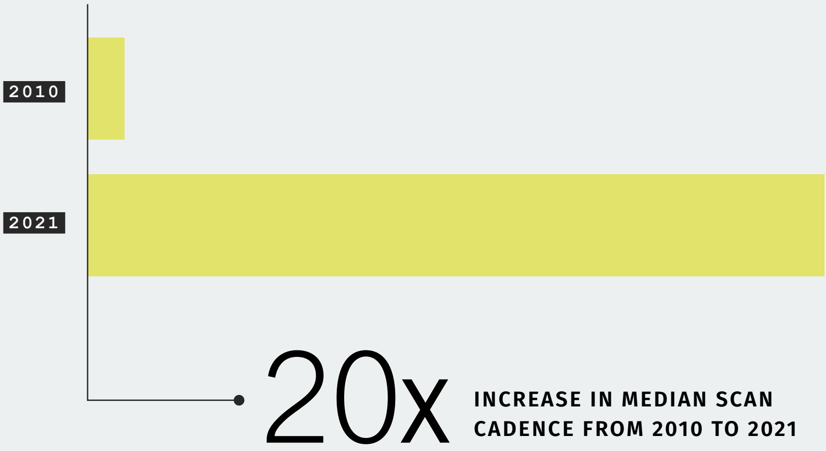
35% of libraries used had a known flaw

10% of libraries used had a known flaw



Scan Cadence

Continuous testing and integration, which includes security scanning in pipelines, is becoming the norm. A decade ago applications were scanned two or three times a year. Now, 90 percent of applications are scanned more than once a week with the majority scanned three times a week.



20x **INCREASE IN MEDIAN SCAN CADENCE FROM 2010 TO 2021**

Multiple Scan Types

We've seen a 31 percent increase in the use of multiple scan types between 2018 and 2021, with much of that gain coming from organizations using the full suite of static, dynamic, and SCA scans.

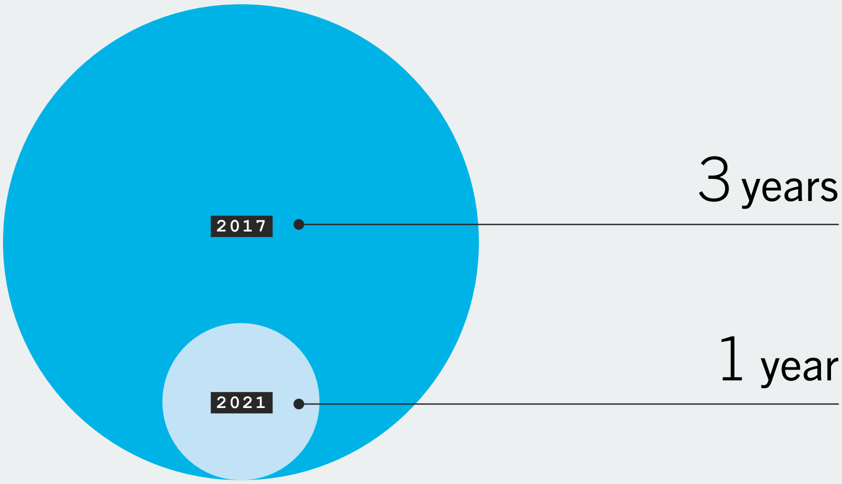
31%

INCREASE in the use of multiple scan types

Third-Party Libraries

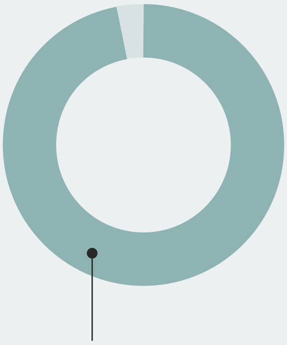
77% *of flaws in third-party libraries remain unfixed after three months*

On a positive note, there is a noticeable improvement in time to remediation for third-party flaws. Back in 2017, it would take over three years to get to the 50 percent (half-life) closed point, and now it takes just over a year.



Open Source

Open-source libraries are still a significant cause for concern.



97%

of Java applications are made up of open source libraries

Veracode Security Labs

On average, organizations with Veracode Security Labs training decrease their time to fix 50 percent of flaws by 35 percent.

35%

DECREASE in the time it takes for organizations to fix flaws

SECTION 02

Introduction

In 2019, for our 10th annual State of Software Security report, we began looking at the specific concerns associated with the use of open-source software and have been fortunate to be able to map the complex landscape of secure software development. We've identified a few ideas that many of our customers probably feel in their hearts, and we confirm them with data — things like scanning at a regular, rapid pace is good.

Security debt can build over time, and addressing it early can help mitigate work down the road. Using multiple types of scanning — static, dynamic, and software composition analysis — can give a fuller picture of an application's security, and it helps remediation happen more quickly and more completely.

These things can help every application, even those old creaky legacy applications, and it's been rewarding to be able to verify and quantify the effect of what many developers feel makes applications more secure.

So where does that leave us for this 12th report?

We feel like we've quantified some of the mysteries about application security with Veracode's extensive data, and we could continue to do that. But we think it behooves an industry to occasionally take a step back to try to get a view of the past and take a look toward the future — to see where the landscape has been steady and where it's changed and to try to understand which principles have stood the test of time and which have faltered.

So we're going to do just that:

1. **Look at the use of software analysis tools**

   We'll start with a look at how people are using software analysis tools and how that's changed over the years. We'll see development trends reflected in those scans. We'll look at how free and open-source software continues to be integral (though variably so) to most applications.

2. **Analyze flaws in software**

   Then we'll look at how those development trends manifest themselves in the flaws that get introduced into software.

3. **Examine how flaws are fixed**

   Next, we'll examine how things are fixed and whether developers are getting better at fixing things.

4. **Look to the future of secure software**

   Lastly, we'll take a peek into the future and think about what exactly developers can do to write more secure software. In particular, we'll see that the simple act of taking time to learn how to fix flaws helps get them fixed faster and helps prevent future bugs from showing up.

Let's take a quick trip down memory lane . . .

SECTION 03

How Software Development Has Changed

One of the advantages of serving the software development community for so long is that Veracode is able to see changes in development practices over time. So rather than diving right into security this year, we want to focus on how developers themselves are approaching applications and how that's changed.

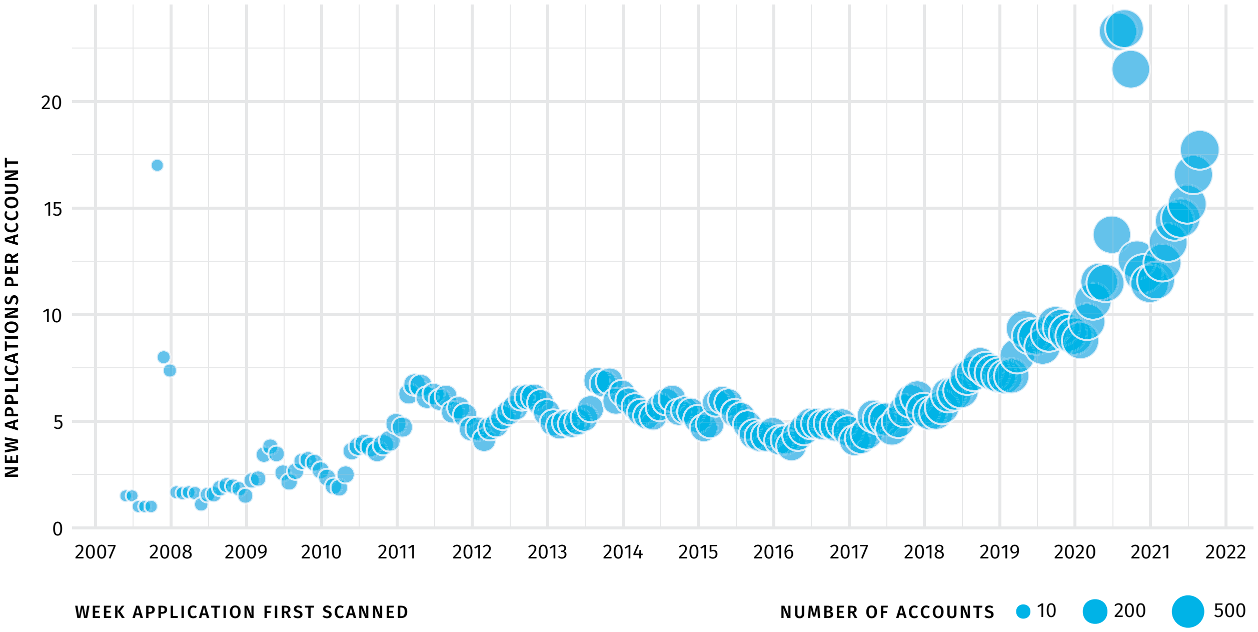
The Number of Applications Scanned Has Tripled

First, we want to examine just how many applications developers are scanning for flaws. Figure 1 shows that more applications are being scanned than ever before. And the increase is not simply due to the fact that there are more organizations. In the last year, most organizations are creating, on average, more than 17 applications for scanning per quarter, up from approximately five a decade ago. But why might this be the case?

We have two hypotheses:

1. Organizations are creating smaller, more modular, applications that do a single thing.
2. Organizations are expanding the scope of their security to lower-criticality applications.

Figure 1: Application creation over time

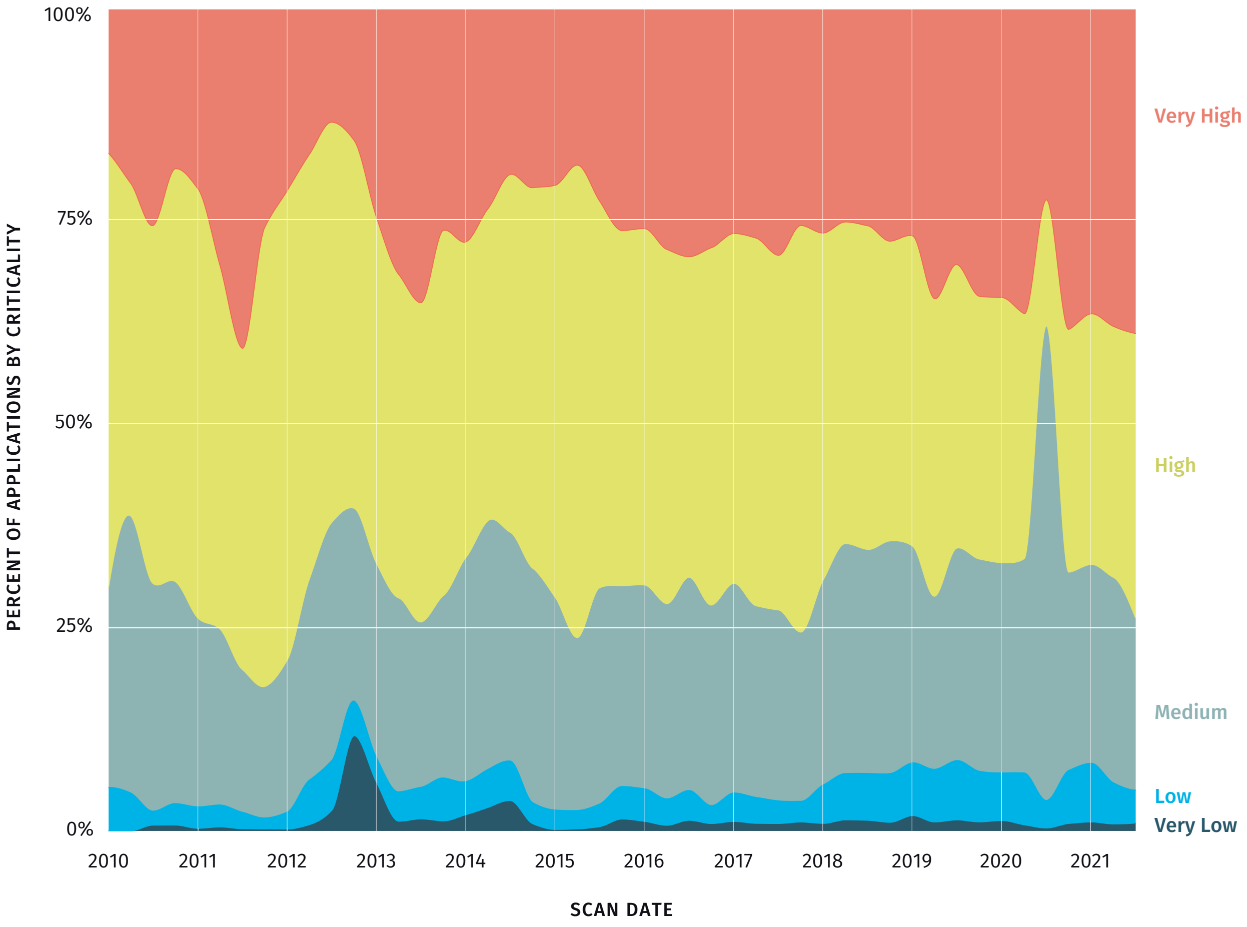


Scanning Average

Organizations are scanning, on average, more than 17 new applications per quarter. This number is more than triple the number of apps scanned per quarter a decade ago.

We actually see that the latter is not true in Figure 2. The distribution of app criticality has been fairly constant, with some bumps along the way when new or existing users onboard many applications (as was the case in mid-2020 when a single user scanned a few hundred "Medium" criticality applications). The skew has been pretty consistent over the last 10 years, with most applications having "High" or "Very High" criticality, and only a handful registering "Low" or "Very Low."

Figure 2: Application criticality over time



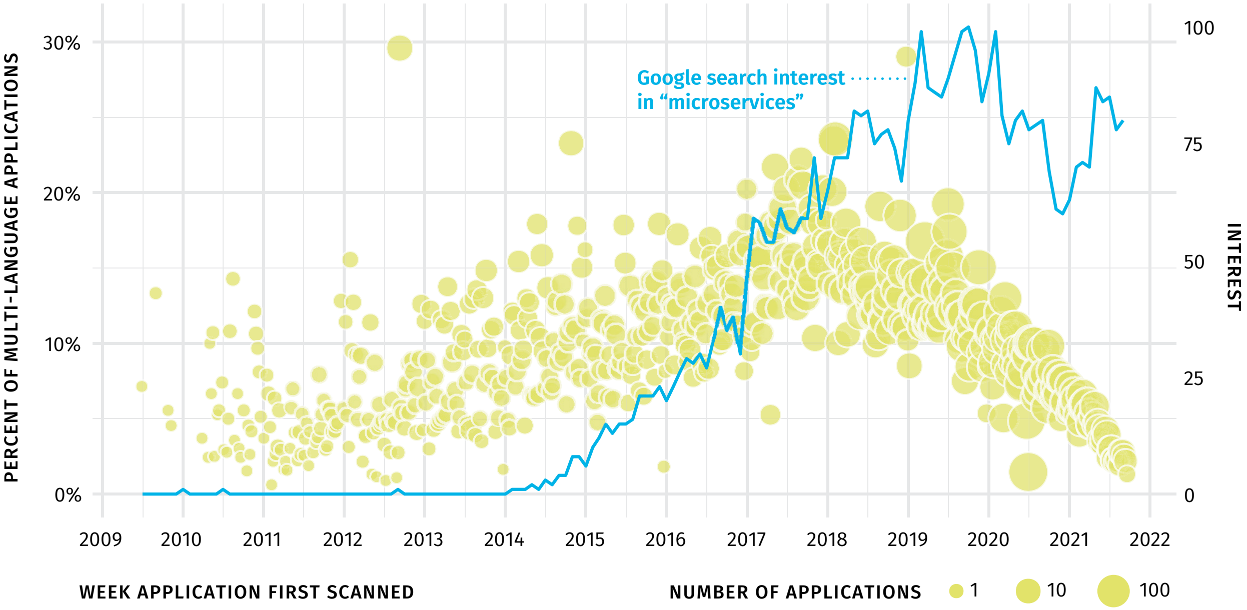
If developers are not simply scanning applications they considered unimportant before, perhaps there is a profusion of new applications — smaller, more modular ones. Some might call them "microservices."

The Rise of Microservices

What defines microservices? They are collections of loosely coupled applications, usually with a small codebase, that communicate via APIs. The advantage of microservices is that it's easier to work on the various parts of an application if changing one part is unlikely to affect the other bits.

So how might we see this reflected among Veracode users? Well, we'd expect applications to increasingly use one language and become smaller in size. Figure 3 looks at the first part of that hypothesis.

Figure 3: Use of multiple languages in new applications



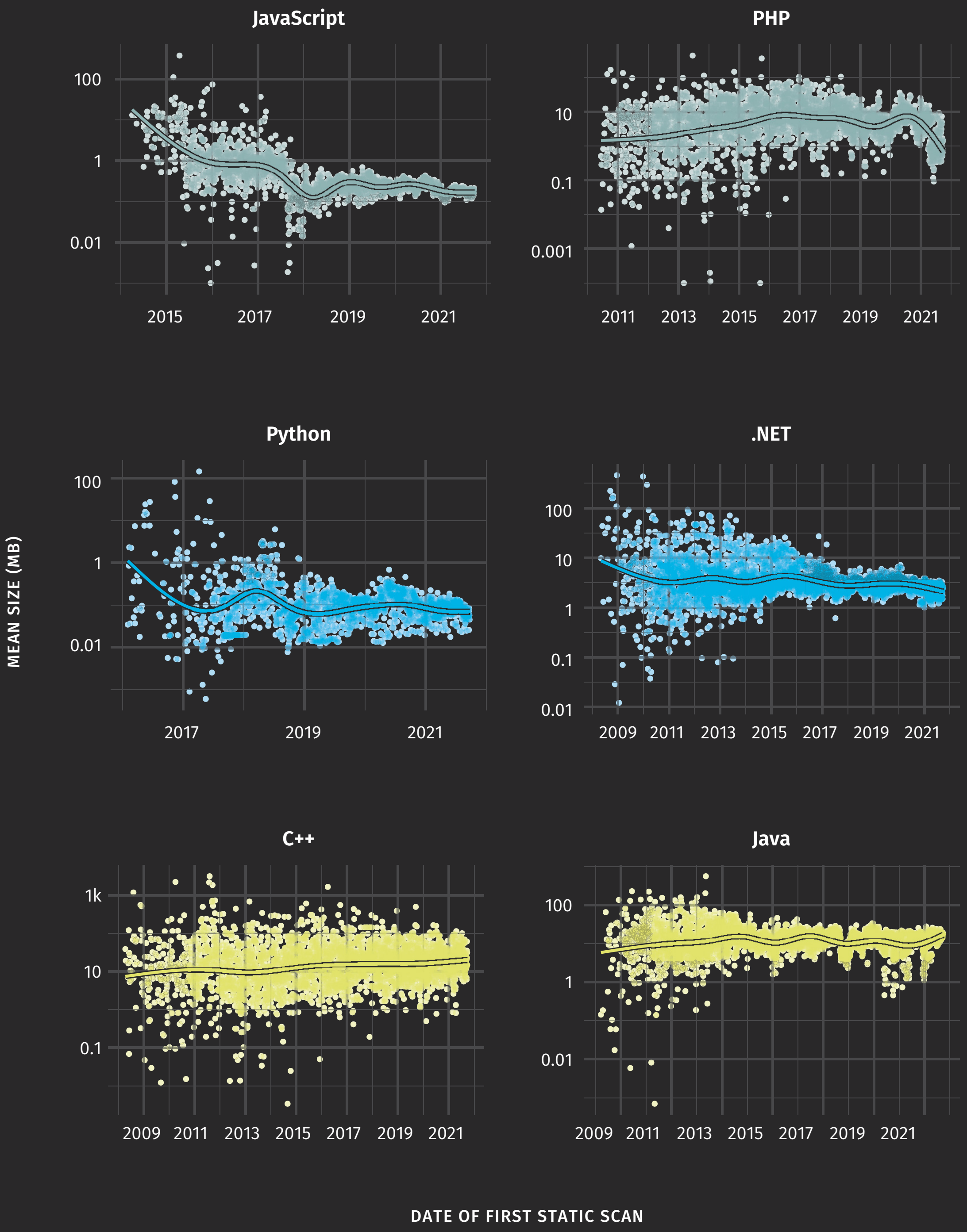
Are developers pivoting to microservices?

In 2018, roughly 20 percent of applications incorporated multiple languages. This year, less than 5 percent of apps used multiple languages.

Up until roughly 2018, there was a slow but steady increase in the number of applications using multiple languages, up to a peak (excluding outliers) of about 20 percent of apps incorporating multiple languages. But as the notion of microservices gained favor and took over, there was a nosedive, with less than 5 percent of applications currently using multiple languages.

So we see that developers are using one language at a time, but are their applications getting smaller? Figure 4a says it's complicated.

Figure 4a: Application size over time



Applications written in a few languages we might consider "good" for a microservice-type architecture certainly have declined in size.

JAVASCRIPT

JavaScript applications have gotten considerably smaller over time, possibly with the inclusion of a more diverse and robust library ecosystem.

PYTHON AND .NET

Both Python and .NET have seen reductions in size, but that may be more regression to the mean than a true trend.

C++ AND JAVA

Meanwhile, applications written in more established languages like C++ and Java have remained more or less the same size over the past few years.

SCALA

Scala applications (not shown) have seen a decline in size, and the popularity of Scala compared to its more heavyweight godfather Java may have something to do with different architectural goals.

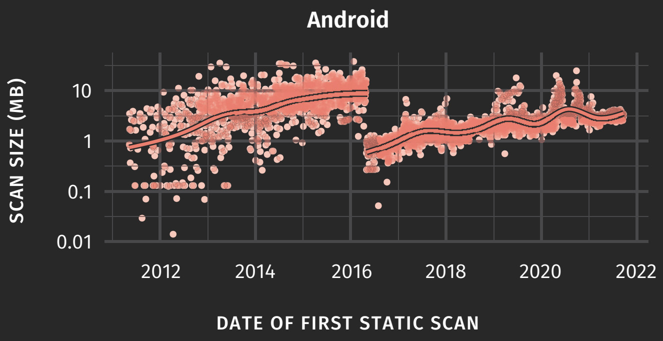
GO

Interestingly, Go (not shown), a language commonly associated with microservices, has actually seen an increase in application size.

ANDROID

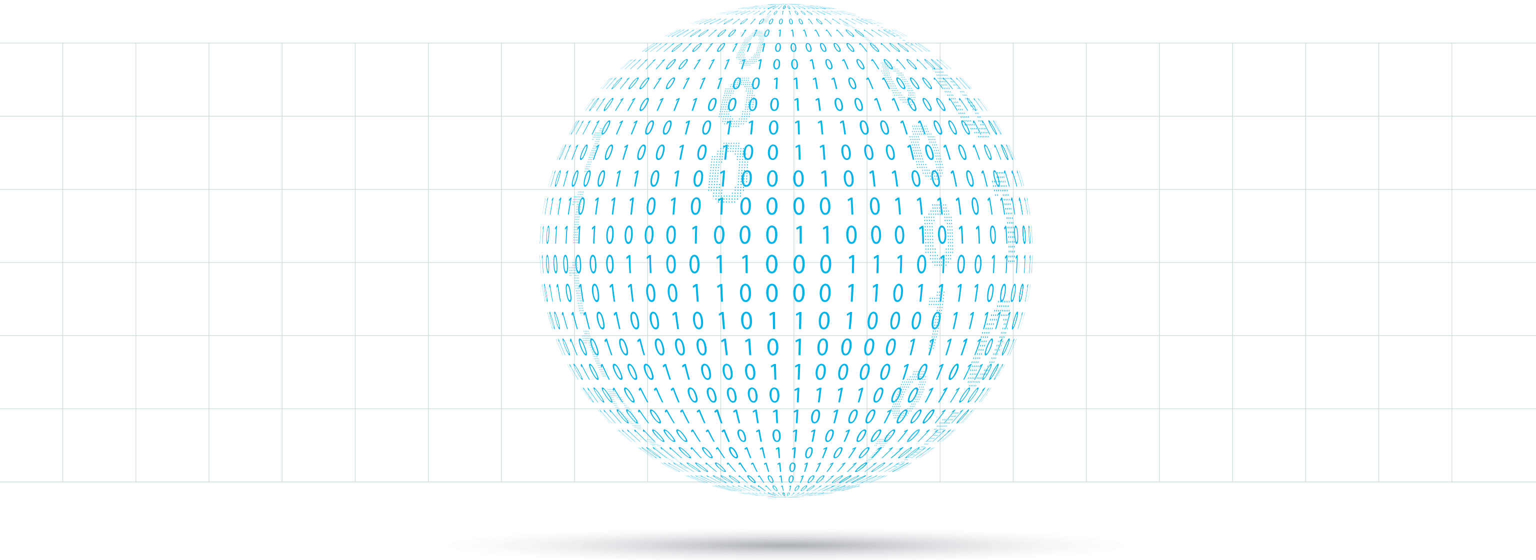
A quick aside on Android applications: Android applications were getting increasingly large until the release of Android N, which switched to an OpenJDK. This allowed for significantly smaller application sizes and was followed by another slow and steady increase. We don't think this is a trend, but it's nice to see major events in software ecosystems validated in data.

Figure 4b: Android's abrupt size shift



Increase in Median Scan Cadence

> **"It is no longer sufficient to scan software as a pre-production step in the last phase of the software development lifecycle. Just as software is now deployed continuously, software security scanning must also happen continuously as a fully integrated part of the software development process."**
>
> **SAM KING, CEO, VERACODE**



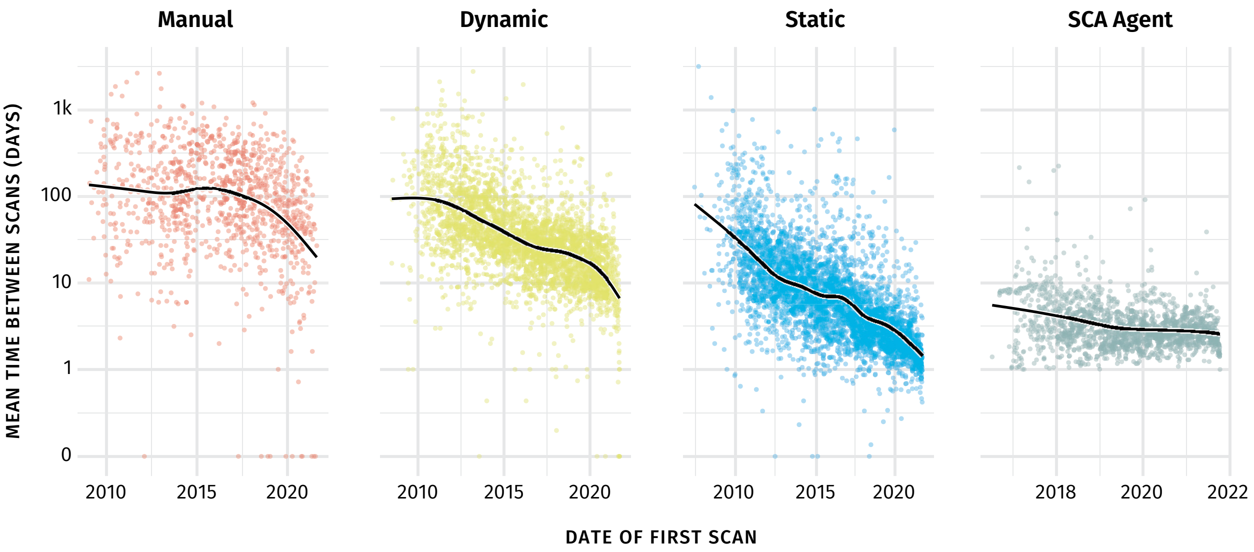
It's been said that "software is eating the world." We think it's probably also fair to say that "agile is eating the software world."

Continuous testing and integration, which includes security scanning into pipelines, is becoming the norm, and we can see that reflected in how often users are scanning their applications. A decade ago users were averaging two or three scans a year. Now, most are running daily static scans and weekly dynamic scans. Software composition analysis (SCA) scans also occur at least weekly. The sooner in the lifecycle you can discover problems, the more likely you'll be able to solve them quickly, before they become bigger a problem down the road.

But is scanning more *good*?

If you look back at SOSS volumes 9, 10, and 11, you'll see that applications that are scanned at a regular cadence fix more flaws faster than those that are only scanned periodically. Security seems to prefer agile development.

Figure 5: Scanning cadence over time



Median Scan Cadence

2010

Median application was scanned less than once a month (only 10 percent of apps scanned more often than weekly)

2021

90 percent of apps scanned more than once a week (majority scanned three times a week)

20X

increase

Organizations Are Using Multiple Types of Scanning

Part of the advantage of the continuous integration paradigm is the ability to easily add new components to the pipeline. Static testing? A must. The use of dynamic analysis is growing as well, and since we're becoming more and more aware of the potential risks inherent in open-source software, it's a no-brainer that secure development includes software composition analysis.

We've seen a 31 percent increase in the use of multiple scan types between 2018 and 2021, with much of that gain coming from organizations using the full suite of static, dynamic, and SCA scans. We're not just pointing this out as a random fact.

LAST YEAR WE FOUND:

Organizations that used dynamic in addition to static scanning were able to remediate

50% of flaws

On average

24 days faster

And including SCA shaves off

another 6 days

So fire up those scanners, folks!

> “In many respects, development teams have shifted from writing software to assembling software.”
>
> **CHRIS WYSOPAL, CTO AND CO-FOUNDER, VERACODE**

Software Bill of Mistakes

Organizations Heavily Leverage Open-Source Libraries

How has open-source, and, more generally, third-party software changed over the last few years?

Last year's report looked at the proportion of code included in each scan that was third-party code versus homegrown. What we saw was interesting. Most applications (depending on the language) had a kind of barbell effect, being composed of almost entirely third-party code or almost entirely in-house code.

There were of course some exceptions. Java's OOP design philosophy of gluing classes together until your code begins to look like a functioning application makes code reuse a breeze. And why write your own classes when there are perfectly good third-party ones freely available? The result is that most of the code in Java applications comes from third parties. But have those barbells evolved over time? Let's take a look at Figure 6.

+ JAVA

Java remains steadfastly mostly third-party code and has pushed even more so in that direction in the last few years.

+ .NET

There is an interesting "shock" to the data for .NET: In mid-2020, we saw an abrupt shift in the percentage of third-party code in .NET applications. The relative time period coincides with the release of .NET 5 (formerly .NET core), which integrated and unified a good amount of functionality into a single framework. It's good to see developers keep up with the latest updates.

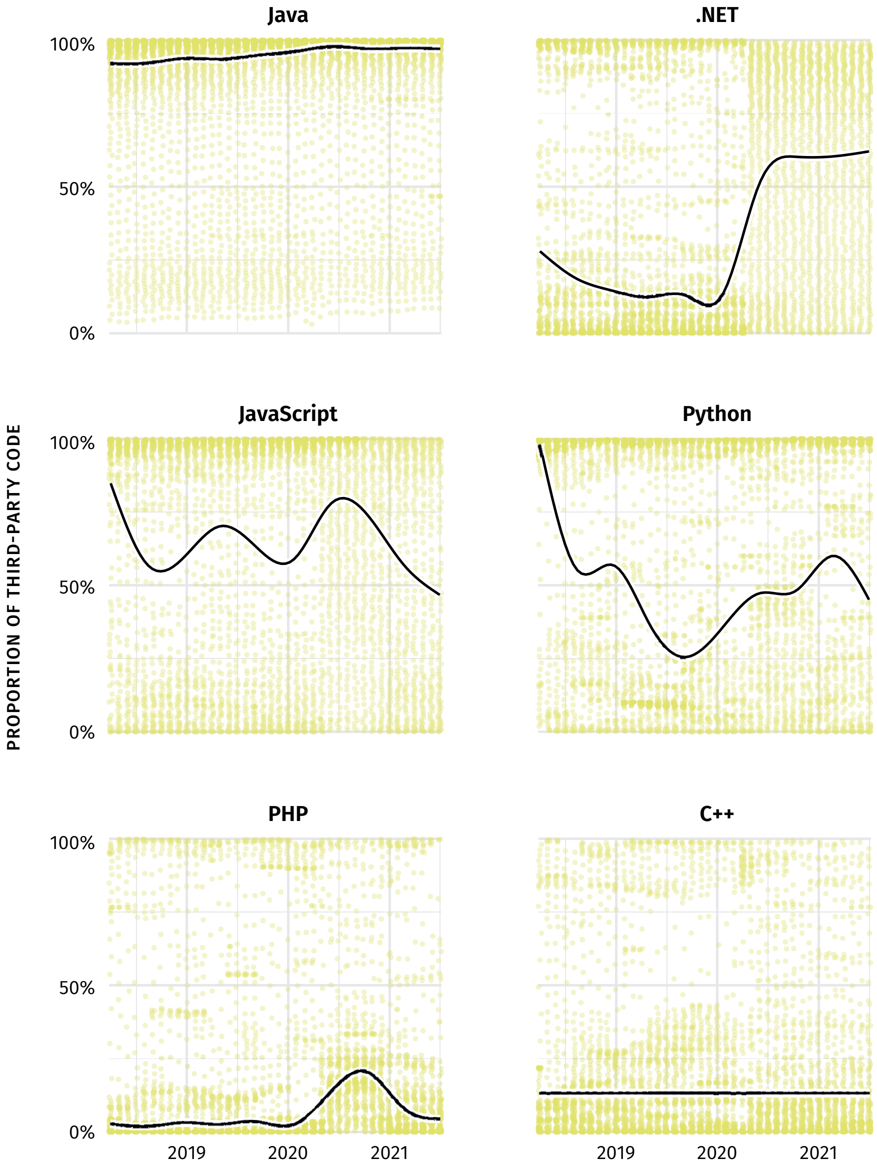
+ JAVASCRIPT AND PYTHON

JavaScript and Python show the barbell effect, with applications being either mostly homegrown or mostly third-party libraries, causing the trend line to bounce around the middle over time.

+ PHP AND C++

PHP and C++ remain relatively constant, leaning heavily toward mostly homegrown code.

Figure 6: Third-party code by language



Most Developers Stick With the Same Libraries Year Over Year

Keep on keeping on . . .

We found that developers stick with tried-and-true libraries and rarely attempt to refactor their code base to pick up the "coolest" or "most-popular" libraries.

We are seeing some evolution in terms of how much third-party code developers are using in each language.

Last year's open source report looked at shifts in the use of vulnerable libraries between 2019 and 2020. Since we're focusing on how things have evolved over as long a time period as the data can muster, we want to expand that view a little bit here. Figure 7 takes a look at how the top 10 most popular libraries across our six languages of interest have evolved over time.[1]

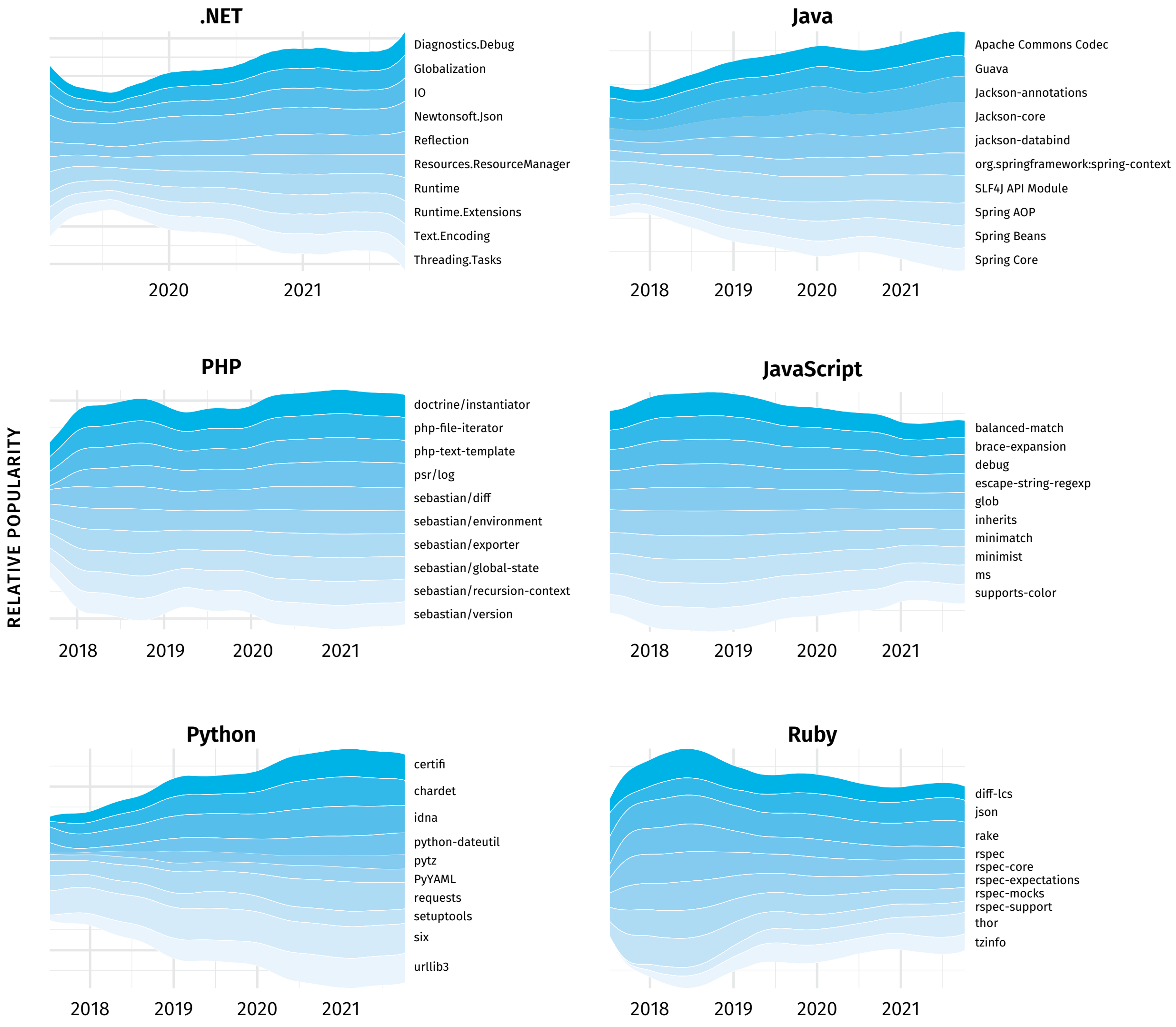
Figure 7 is what is referred to as a stacked area chart. Each band represents the percentage of scanned repositories that are using a particular library. The thicker the band, the more popular the library. When the overall height of the chart increases, it indicates that those 10 libraries are all growing in popularity. If libraries waxed and waned in popularity frequently, we would expect to see large swells of color that might peter out over time. We don't see that in Figure 7, but rather mostly uniform popularity of those top libraries.

We won't spend a ton of time going over each wiggle. But what we do see is that, for all the languages, the most popular libraries haven't changed all that much. Things like debug and inherits continue to be popular for JavaScript though they may swap around the top few spots year over year. The larger lesson here is that developers are going to stick with tried-and-true libraries and likely aren't going to attempt to refactor their code base to pick up the latest hot commodity.

This is indeed what we saw in last year's SOSS: *Open Source Edition*. Updates happened slowly when libraries didn't have flaws, but relatively quickly when they did. As long as open-source developers continue to fix security flaws, developers will keep on using those libraries.

[2] A few notes on this figure. We are swapping Ruby in here for C/C++ as most C/C++ applications don't use an explicit package manager and instead use makefiles to see if appropriate libraries are present. This can make extracting what applications are using what libraries pretty tough. We only have data going back to 2019 on .NET, so be sure to check the scales. Finally, the colors in this figure are used to differentiate the individual libraries, but are reused for purely aesthetic reasons.

Figure 7: Popular libraries by language



Third-Party Libraries Have Fewer Flaws

We've seen library usage evolve over time, though maybe not in a nice smooth trend. But what implications does that have for security? After all, that's really what we're here to talk about, right? Making applications more secure. Even the federal government is taking notice of this whole "third-party libraries might be risky, actually" idea.

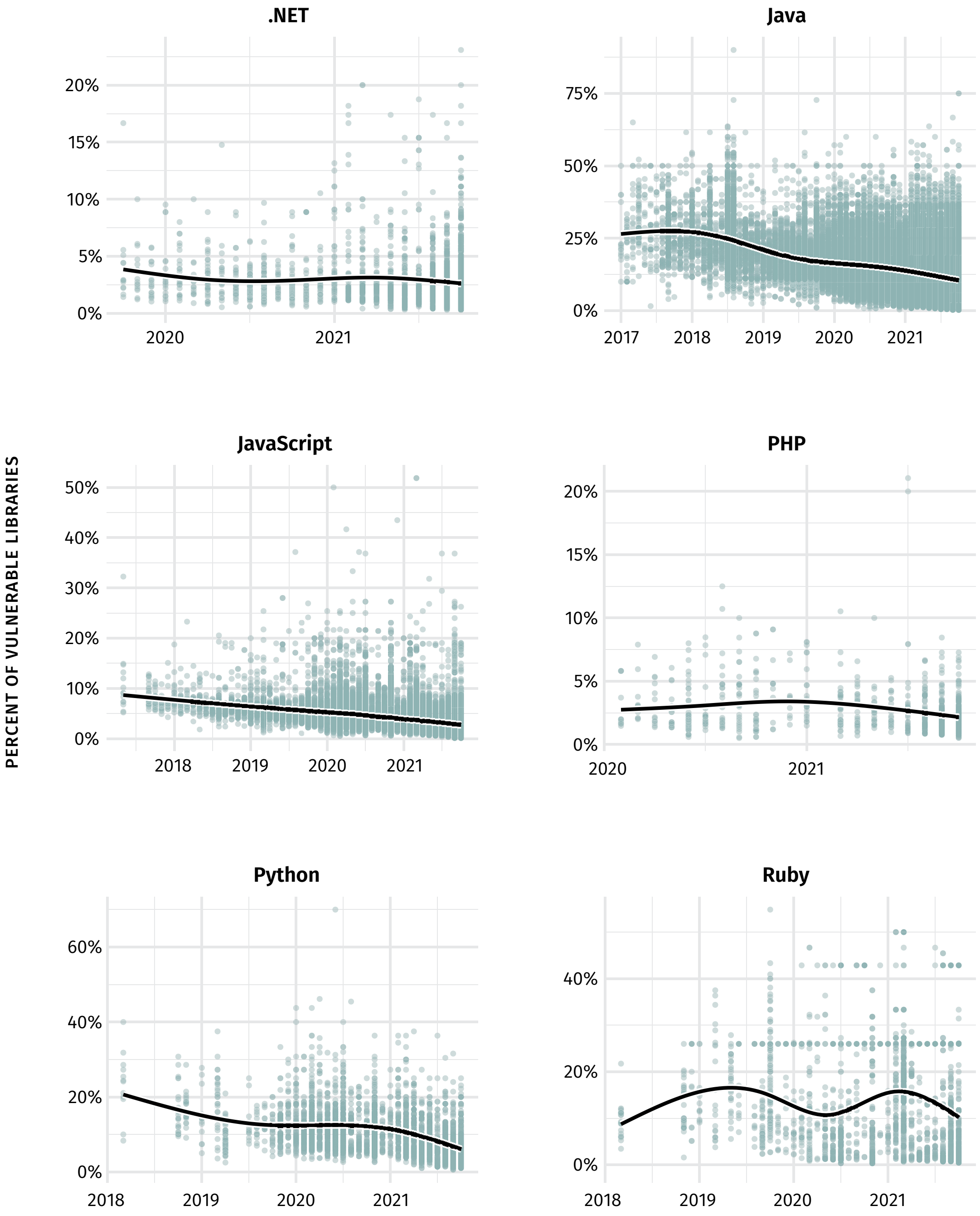
The recent Executive Order on Improving the Nation's Cybersecurity lays out the following about securing the software supply chain:

"The development of commercial software often lacks transparency, sufficient focus on the ability of the software to resist attack, and adequate controls to prevent tampering by malicious actors. There is a pressing need to implement more rigorous and predictable mechanisms for ensuring that products function securely, and as intended. The security and integrity of 'critical software' — software that performs functions critical to trust (such as affording or requiring elevated system privileges or direct access to networking and computing resources) — is a particular concern. Accordingly, the Federal Government must take action to rapidly improve the security and integrity of the software supply chain, with a priority on addressing critical software."

So are applications using more or fewer flawed libraries? As with many things in our research, Figure 8 tells a language-specific story. (But before you dive into Figure 8, notice that the vertical axes are different for different languages.)

There are clear trends for Java, JavaScript, and Python, and that trend is very good, because it goes steeply down. In 2017 nearly 35 percent (on average) of libraries used had a known flaw. In more recent years that has come down to nearly 10 percent. JavaScript has gone from about 10 percent to less than 4 percent, Python from about 25 percent to nearly 10 percent, and Go (not shown) from 7 percent down to 4 percent.

Figure 8: Percent of flawed libraries by language



SECTION 05

This type of report would be relatively easy to create if all — or heck, even some — of the attacks were fresh and new. The stories would almost write themselves. But history is teaching us that we will experience the same types of flaws year after year. Sure, there are variations among languages and things may shift around in prevalence. But by and large, the technical flaws themselves don't go away, and any changes we do observe tend to evolve slowly.

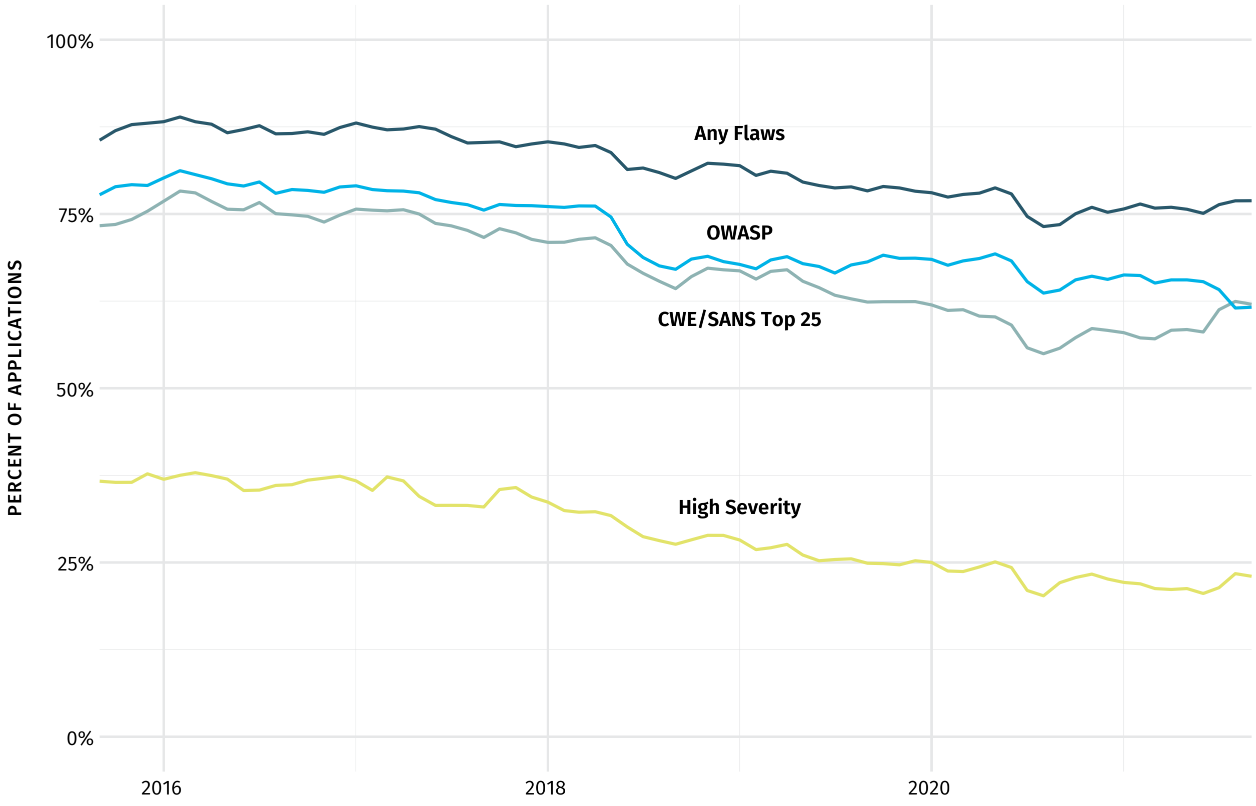
The Flaws of Yesterday Are (Still) the Flaws of Today

Take as an example Figure 9 (or really any of the upcoming plots). We pulled out the flaws listed in the OWASP Top 10 and CWE/SANS Top 25 and those classified as "High" criticality or above. If you look closely at each of those over time, you'll notice some peaks and valleys. But we want you to pull back a bit, and perhaps squint your eyes so that you can see the overall trend in these plots. Notice that, even though the lines may bounce around, they are all slowly decreasing.

That's good news: The trend across all the applications is a general reduction in flaw prevalence.

The trend across all the applications is a general reduction in flaw prevalence.

Figure 9: Percent of applications with various flaw types in static analysis

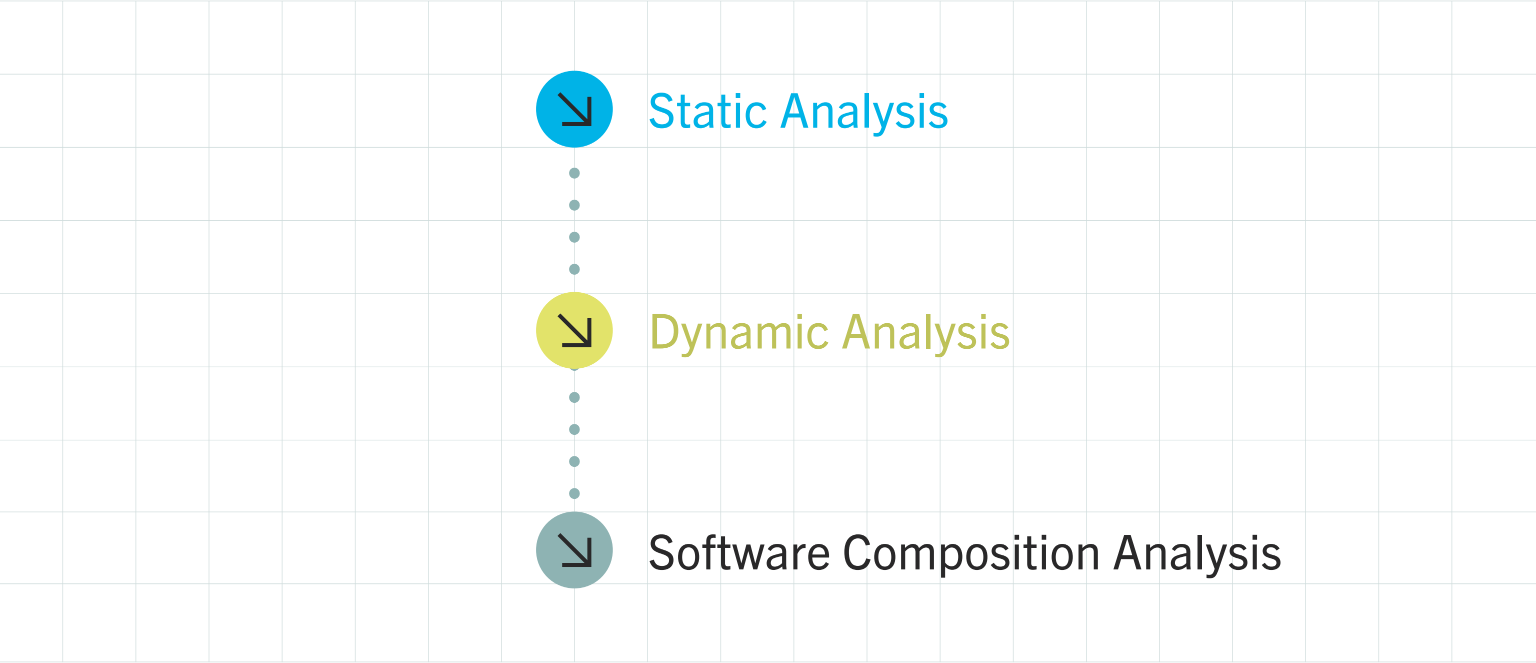


The Lowdown on Static, Dynamic, and Software Composition Analysis

Even though we just said you should take a step back and look at the big picture, we offer this next plot as an example of both why that's a good thing and why you don't want to stop there.

As you look at the rankings in Figure 10 you'll probably notice that CRLF injection is more prevalent than information leakage. Maybe you find that helpful, but as anyone who's worked in more than one language will tell you, each language has its own strengths and weaknesses, and this applies to secure development as well. The development language matters when it comes to the types of flaws introduced — at least for some of the detection methods.

Therefore, it's worthwhile to understand a bit about these methods because it will help point us in the direction of further areas to investigate.

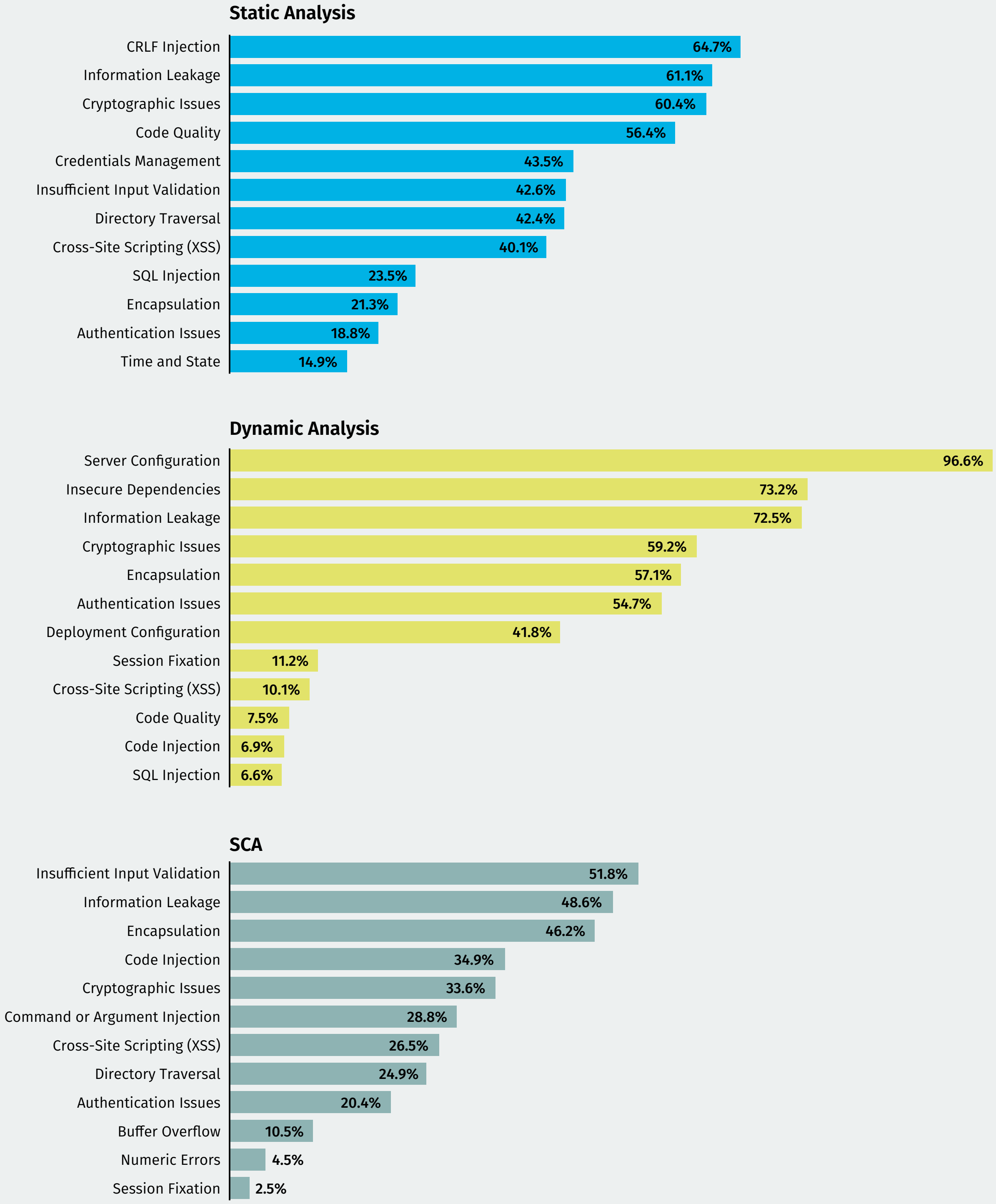


Static Analysis

Dynamic Analysis

Software Composition Analysis

Figure 10: Top software weaknesses discovered by scan type



Static Analysis

Static analysis looks directly at the source code and thus needs to know what to look for in different languages. It's very strong at detecting places where maybe memory isn't managed correctly or input isn't properly validated or sanitized.

And given that, it shouldn't come as a surprise that the results of static analysis are very dependent on the development language. It's rather common to find flaws with buffer/memory management in a language like C++, but those are nonexistent in languages like .NET or Java, where those functions are abstracted away from the developer. So even though CRLF injection is the top flaw type for static analysis overall, it's not even in the top 10 flaws in C++ or PHP.

Figure 11 expands the list of static analysis flaws (the top graph in Figure 10) in two different ways. First, it breaks out the flaws by language, so the differences across languages become more apparent. (But note that the vertical axis differs for different languages.) Second, it adds the element of time. This enables us to see the trends and, again, if we pull back a bit and squint through the details, it's clear that most everything is trending down over the past few years.

There are a few things worth calling out here that are not trending down, though.

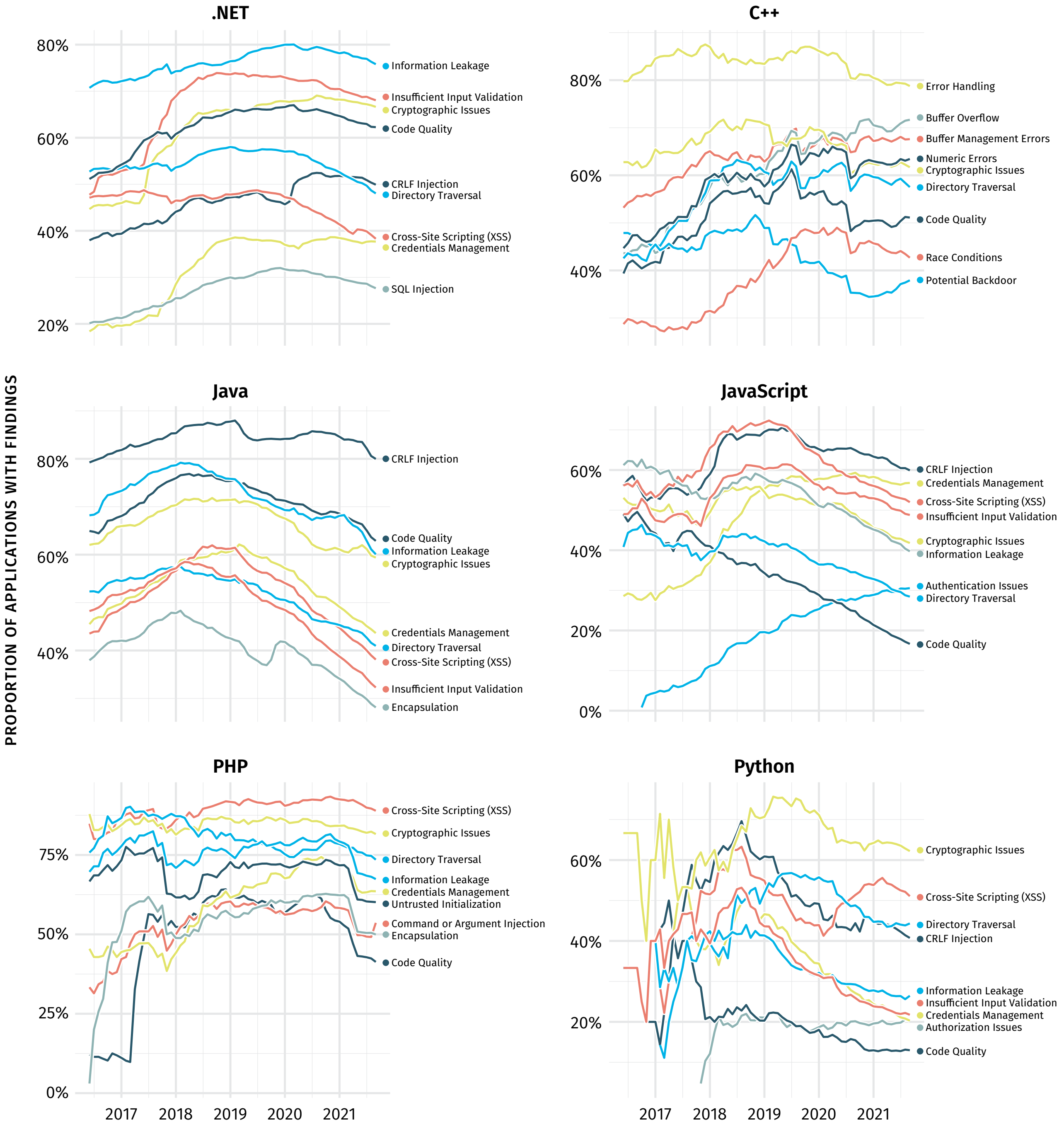
+ **JAVASCRIPT**

  Has some growing challenges with identity management: Both credentials management and authentication issues are trending upward over time.

+ **JAVA**

  And the overall declines across Java are clearly more pronounced than the very subtle declines seen in PHP applications.

Figure 11: Software weaknesses found by static analysis by language

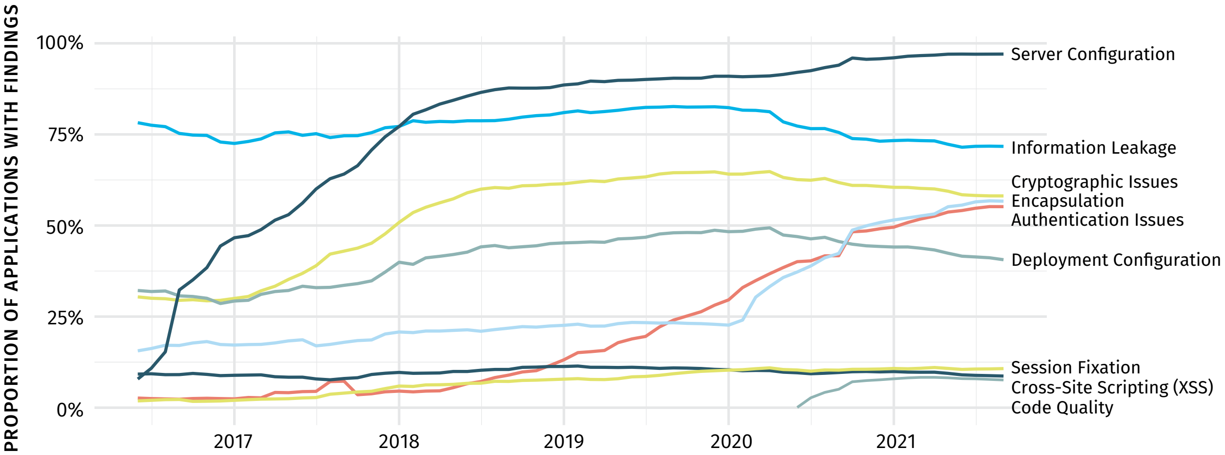


Dynamic Analysis

Dynamic scanning takes advantage of, and is run against, the runtime environment.

It doesn't dive deeply into the idiosyncrasies of the underlying languages but instead can find more flaws in the execution and interface of the code. Notice the top types of flaws here generally don't overlap with the static analysis findings. Server configurations and information leakage were consistently the leading categories of flaws found across all the underlying languages. The flaws were so consistent across languages that it's not worth showing the differences. Each individual language looked like the overall trends shown in Figure 12.

Figure 12: Software weaknesses found by dynamic analysis

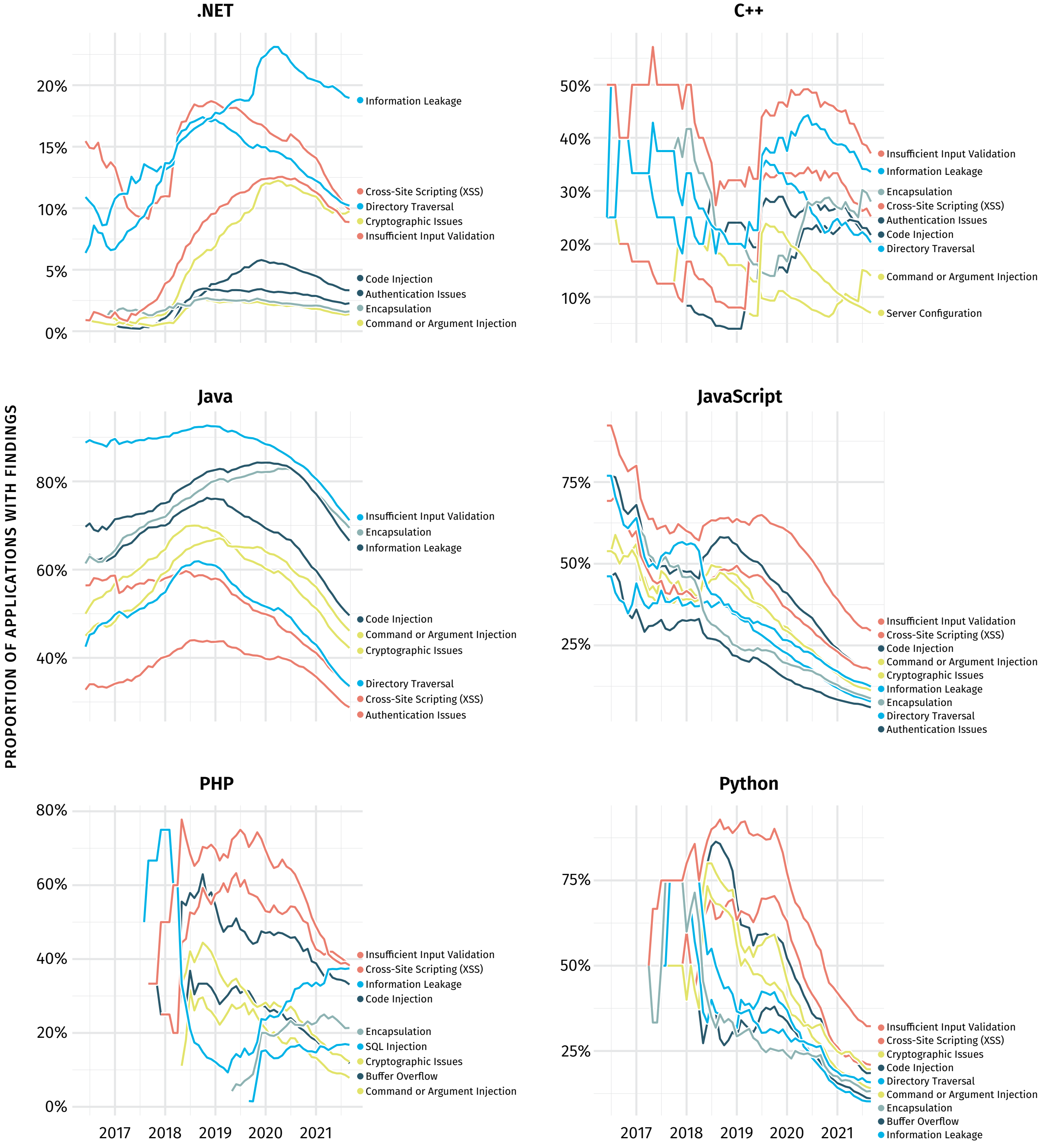


Software Composition Analysis

Software composition analysis is the third type of scan, and it operates by tracking the various open-source projects and packages and then identifying which are included in the code base.

This allows all the existing information in those libraries to be shared with the developers. The flaws in third-party software can be reported from a variety of sources such as static code scans, manual code review, security researchers reporting flaws, etc. Once again, we find that these types of flaws vary by language, as you can see in Figure 13 (but note that the vertical axis uses different scales for different languages). Some languages (Java, JavaScript, and Python) show clear declines while .NET and C++ don't appear to show that type of decline in their third-party libraries.

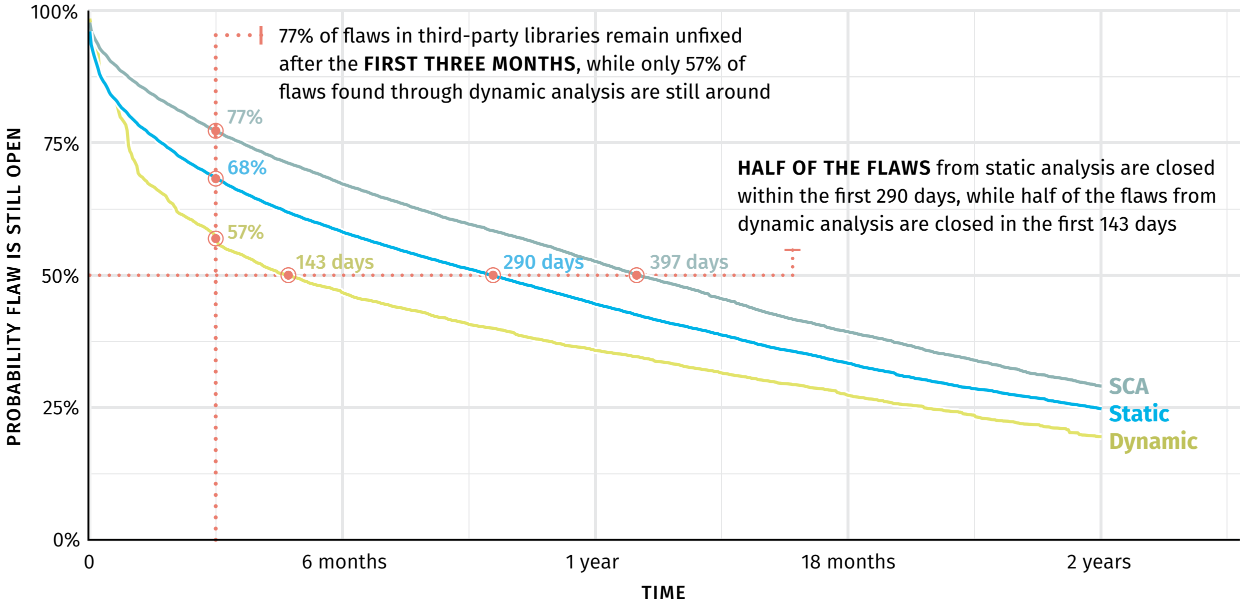
Figure 13: Software weaknesses found by SCA, by language



Fix Rate Comparisons by Scan Type

We've spent our time up until now looking at applications, detection methods, types of flaws and their prevalence, and how developers have changed their security practices over time. But what about actually fixing things? This is a good point to transition into thinking about how many flaws are getting fixed and how quickly (or not so quickly). Anyone who's been reading our research for the last few years will surely recognize this next plot. It looks at the millions of flaws we've been tracking and shows estimates for how long any particular flaw will remain open. It factors in both the flaws that have been remediated and those that have yet to be fixed to give us a more accurate measurement of expected closure rates over time.

Figure 14: Probability of flaw remaining open by scan type



77% of flaws in third-party libraries remain unfixed after the **FIRST THREE MONTHS**, while only 57% of flaws found through dynamic analysis are still around

HALF OF THE FLAWS from static analysis are closed within the first 290 days, while half of the flaws from dynamic analysis are closed in the first 143 days

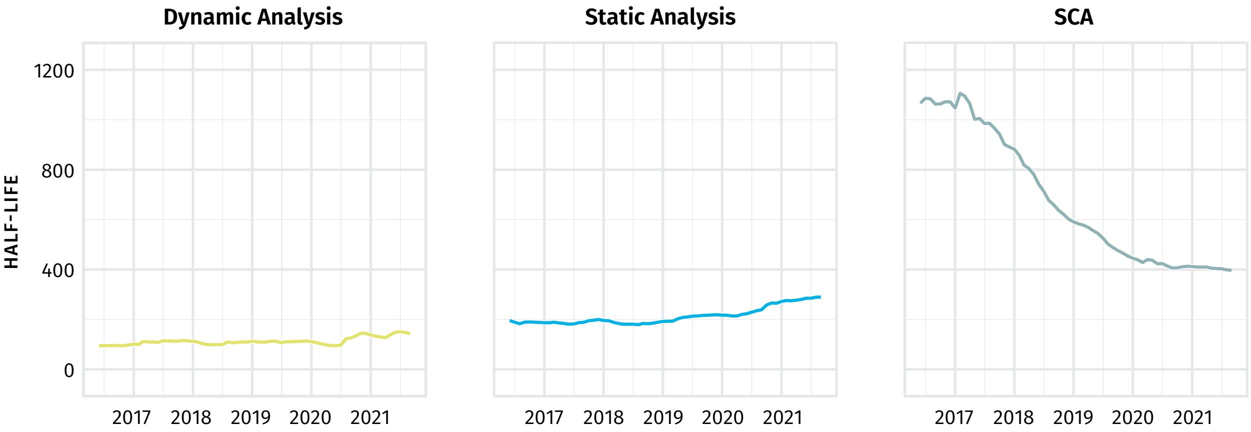
Dynamic flaws get fixed the fastest and SCA flaws the slowest, with flaws from static analysis being fixed at a rate between those two.

This year's research expands beyond just remediations for flaws found through static analysis (we focused on that in previous years) and looks at flaws discovered through dynamic analysis and SCA. Figure 14 shows that dynamic flaws get fixed the fastest and SCA flaws the slowest, with flaws from static analysis being fixed at a rate between those two.

Let's focus on the horizontal dashed line at 50 percent shown in Figure 14. It represents how long it may take to close about half the flaws, or what we can call the "half-life" of a flaw.

It may seem absurd that it can be over a year before even half the flaws identified through SCA are closed — especially when we see flaws found through dynamic analysis lasting just under 5 months for the same half-life metric. But what if we told you that's actually a great improvement? Take a look at Figure 15, which tracks that half-life metric over time. With that historical perspective, flaws found in SCA show a dramatic improvement. Back in 2017, it would take over three years to get to the 50 percent (half-life) closed point, and that's been driven down to what we see today: just over a year.

Figure 15: Half-life by scan type



Remediation efforts have slid back a bit for static scans. In 2017, half-life to remediation was just under 200 days, now remediation has slowed down to just under 300 days.

After looking at the SCA flaws and feeling optimistic that things are improving, take a look at the static scans. It looks like remediation efforts have slid back a bit from an all-time low around 2017 with a half-life of just under 200 days. Currently, remediation has slowed down to just under 300 days. Part of the challenge here is that the competitive market demands timely releases, and software development teams are forced to make difficult tradeoffs between a timely delivery and risk. Some flaws may be left unaddressed to meet deadlines.

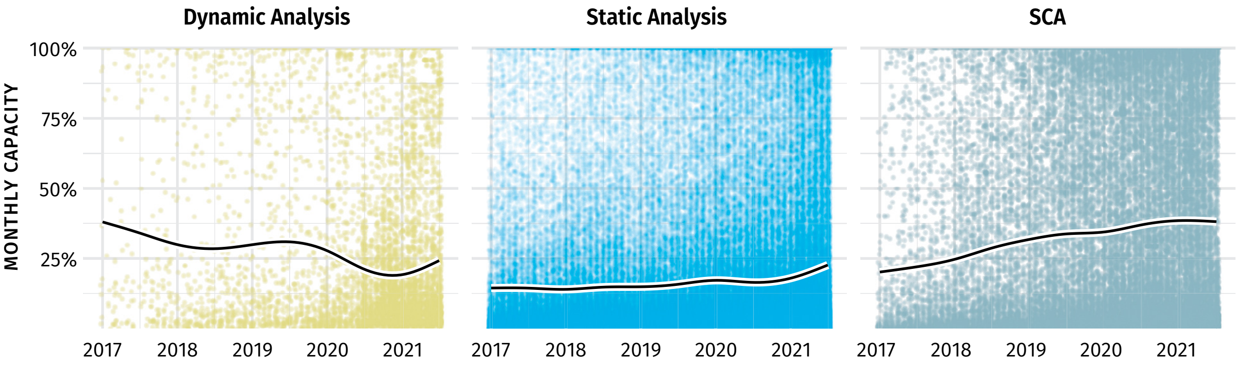
If we want to track speed to remediation, half-life is a good metric, but there is a challenge in relying on it as a complete measure: It doesn't account for all the flaws being fixed in a given time period. In order to measure just how many flaws are being fixed in a window of time, we must look at the overall capacity that development teams have for fixing flaws.

Capacity for Flaw Remediation by Scan Type

To gauge capacity, we look at all the flaws facing a development team in a given month and then see how many of those are fixed in that month. Naturally, there are variations from application to application, but Figure 16 captures the overall trend over the last five years.

The points in Figure 16 represent individual applications, and the lines show the overall trend. Keep in mind that capacity here is looking at the number of remediated flaws out of the total number of open flaws in a given month. This gives us the percentage of flaws being fixed every month. It's good to see the remediation of both static flaws and SCA flaws in any given month on the rise here even if the increase isn't dramatic and (according to Figure 15) it takes a little longer. The remediation of flaws found with dynamic analysis has been bouncing around a bit, but we should see it stabilize as we get more data.

Figure 16: Monthly capacity for flaw remediation by scan type



Overall, the signs here give us hope that there is a focus on application security and that added attention is having a positive impact on the security of our applications.

SECTION 06

Where Do We Go From Here?

As we wrap up this whirlwind tour of the evolution of software security over time, let's take a look at the future and think about what practices might be next in improving application security.

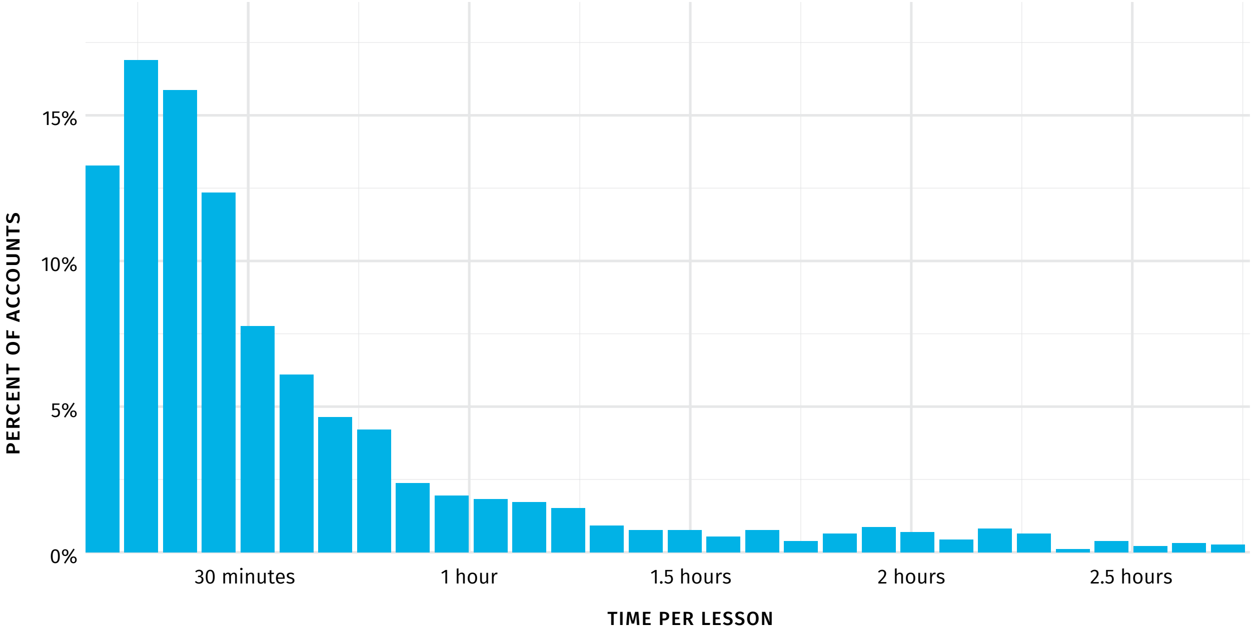
Most Organizations Using Veracode Security Labs Are Fixing Flaws Faster

Here at Veracode, we don't want to simply point out, "Hey, there's a flaw here, you should fix it" — we also want to help developers better understand how to fix those flaws and avoid creating new ones in the future. To that end, we've established Veracode Security Labs to give developers hands-on experience fixing common flaws.

Before you roll your eyes, we want you to know this isn't just a pile of training videos that developers are expected to slog through. Veracode Security Labs lessons are fully realized example applications, written in a variety of languages, with real flaws. The lessons help developers understand the flaws by giving them hands-on experience actually exploiting them. Using real code, developers are led through examples of specific coding flaws. They then develop and execute exploits to build their intuition about security flaws. And more importantly, developers write the patches that fix the flaws, giving them valuable experience when they are alerted to flaws in their own code.

This is starting to sound like a marketing pamphlet, but we promise it's not. We want to know if this type of system can help developers fix flaws faster and help prevent the creation of flaws in the future. Most lessons are short, averaging less than an hour to complete.

Figure 17: Time spent learning in Veracode Security Labs

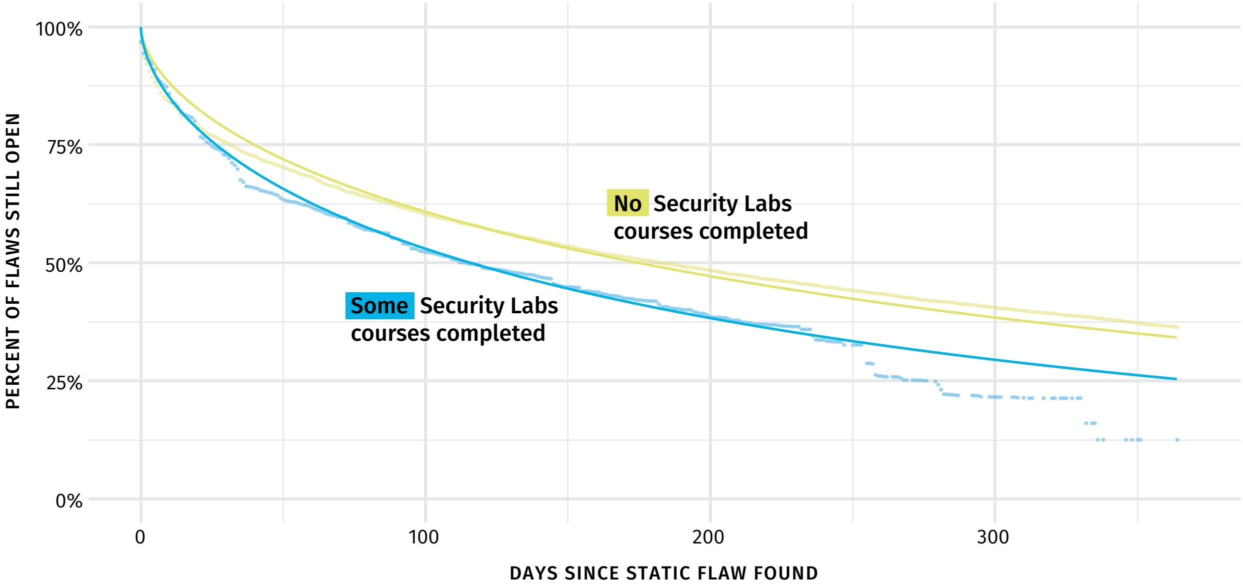


But do these efforts actually make a difference in the ability to fix flaws?

We looked at organizations that had completed some number of lessons and compared the flaws that were found in their applications before the lessons were completed to those that were found after.

Good news: Figure 18 indicates that flaws found after Veracode users completed at least one lesson were fixed faster than those found when developers had no such training. Specifically, 50 percent of flaws were fixed within approximately 110 days by those with training, and 170 days without — a two-month difference on average!

Figure 18: Probability of flaw remaining open by training history



Organizations with Veracode Security Labs Training

110

days it takes to fix approximately 50% of flaws

Organizations without Veracode Security Labs Training

170

days it takes to fix approximately 50% of flaws

2 month difference

So Veracode Security Labs training helps developers fix flaws more quickly, but does it prevent the creation of new ones?

To examine this:

1. We narrowed things down to users who had scanned applications both before and after completing our e-learning courses.
2. We then examined what fraction of scans had flaws.

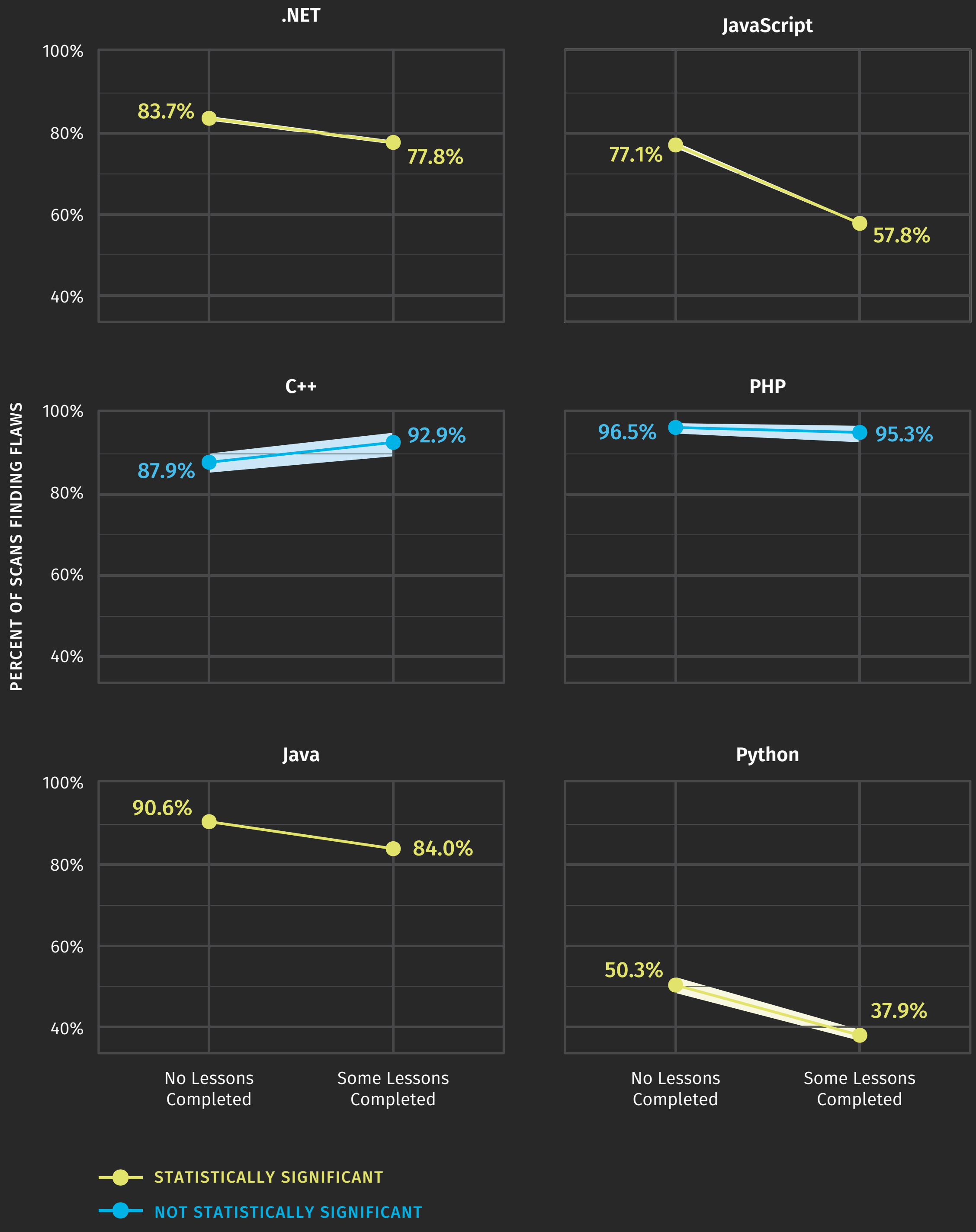
Figure 19 shows that for most languages, the percentage of static scans that find flaws is reduced after some lessons are completed. For JavaScript, this reduction is nearly 20 percent, while other languages see a more modest reduction. There seems to be a slight increase for C++, but it should be noted that it is not statistically significant, nor is the small decline in PHP.

These results don't point to this kind of training as a security cure-all, but they do indicate that training benefits developers. This might seem obvious: If developers have an opportunity to see flaws in a safe, guided environment, they'll be better equipped to fix things "in the wild." But for research reports like this, we don't like to call anything "obvious" until we've seen the data.

Of course there are some caveats to these early results. Given that an organization likely has some inclination toward secure development to even engage Veracode, a larger effect might be seen for developers who never considered security scanning before. We are also painting with an extremely broad brush here.

In the future, breakdowns of the types of lessons and the types of real flaws fixed would give us a better understanding of where this type of hands-on experience works best.

Figure 19: Flaws found before and after training



SECTION 07

Conclusions

In some ways, it's staggering to look back at the history of software development through the lens of Veracode's data. For some of the results in this report, we were able to look back nearly 16 years to the time the first applications were scanned by Veracode customers. As new best practices, different threats, and better capabilities have been developed, we've gotten to take a look at more detailed versions of the evolution of application security.

So standing here in 2022, what can we say we've learned?

Agile development of small, modular applications has eaten the world.

We've seen an explosion in the number of applications being scanned. Many are single languages, and applications in certain languages are getting smaller. Maybe those folks at Bell Labs had some good ideas back in the late 1970s.[2] We've seen developers move from scanning their applications once a quarter to once a day, as well as expand their use of different scanning technologies.

Given that we know that more scanning using multiple tools means faster fix times and less security debt,[3] this shift can only be viewed as good for the future of application security. It's yet to be seen whether the pendulum of history will swing back to monolithic applications and waterfall development, but that doesn't seem likely now.

Free and open-source code will continue to be a blessing and a curse for developers.

We see no signs that the use of third-party libraries has changed dramatically, or even the libraries developers are using. Developers appear to be using fewer libraries with known flaws, and that's cause for optimism.

[2] The philosophy of the UNIX operating system has been highly influential throughout computing. Some innovations it can take credit for are things like hierarchical file systems, multi-tasking, and multi-user systems. Here we refer to the idea that systems should be composed of small (even trivial), interchangeable programs that can easily communicate. When these programs are composed they are capable of exceedingly complex tasks that might otherwise be difficult to achieve in a monolithic program. We are seeing a reimagining of this idea here, with developers changing focus from large monolithic applications that do a wide variety of tasks to smaller composable parts. Read more: Ritchie, Dennis M., and Ken Thompson. "The UNIX time-sharing system." Bell System Technical Journal 57, no. 6 (1978): 1905-1929. Kernighan, Brian W., and John R. Mashey. "The UNIX™ programming environment." *Software: Practice and Experience* 9, no. 1 (1979): 1-15.

[3] Veracode *State of Software Security*, volumes 9, 10, and 11.

Applications are, slowly but surely, getting more secure.

What is perhaps most heartening throughout this analysis is that, nearly across the board, we've seen steady progress toward more secure applications. While some subsets of flaws have increased in prevalence over time, the trend has generally been downward. This is impressive, given that the capacity for and speed of fixes hasn't necessarily increased. We're hopeful this trend will carry on and that the future will continue to look bright.

New tools will continue to help improve the application security landscape.

In the past, we've noted that using different types of scanning means that developers will fix all types of flaws faster and more completely. Having these types of tools built into continuous integration pipelines and IDEs will only speed developer adoption. In addition to that, tools like Veracode Security Labs lessons may be even more impactful in helping developers better understand the origin of security flaws and how to fix them or prevent them from showing up in the first place.

Appendix: Methodology

Our methodology for data analysis diverged slightly from that used for earlier volumes of the *State of Software Security*. In previous years, we would specifically focus on applications that were under active development from a 12-month window.

This year, we wanted to get a longer-term view, so the core data represents the full historical data from Veracode services and customers.

This accounts for a total of:

- *592,720 applications that used all scan types*
- *1,034,855 dynamic analysis scans*
- *5,137,882 static analysis scans*
- *18,473,203 software composition analysis scans*

All those scans produced:

- *42 million raw static findings*
- *3.5 million raw dynamic findings*
- *6 million raw software composition analysis findings*

The data represents large and small companies, commercial software suppliers, software outsourcers, and open source projects.[4] In most analyses, an application was counted only once, even if it was submitted multiple times as vulnerabilities were remediated and new versions uploaded.

For software composition analysis, each application is examined for third-party library information and dependencies. These are generally collected through the application's build system. Any library dependencies are checked against a database of known flaws.

The report contains findings about applications that were subjected to static analysis, dynamic analysis, software composition analysis, and/or manual penetration testing through Veracode's cloud-based platform. The report considers data that was provided by Veracode's customers (application portfolio information such as assurance level, industry, application origin) and information that was calculated or derived in the course of Veracode's analysis (application size, application compiler and platform, types of vulnerabilities, and Veracode levels — predefined security policies based on the NIST definitions of assurance levels).

[4] Here we mean open source developers who use Veracode tools on applications in the same way closed source developers do. This is distinct from the software composition analysis presented in the report.

A Note on Mass Closures

While preparing the data for our analysis, we noticed several large single-day closure events. While it’s not strange for a scan to discover that dozens, or even hundreds, of findings have been fixed (50 percent of scans closed fewer than three findings; 75 percent closed fewer than eight), we did find it strange to see some applications closing thousands of findings in a single scan.

Upon further exploration, we found many of these to be invalid. These large collections of flaws were both added and removed in single scans: Developers would scan entire filesystems, invalid branches, or previous branches, and when they would rescan the valid code, every finding not found again would be marked as “fixed.”

These mistakes had a large effect: The top one-tenth of 1 percent of the scans (0.1 percent) accounted for almost a quarter of all the closed findings.

These “mass closure” events have significant effects on measuring flaw persistence and time to remediation and were ultimately excluded from the analysis.

01

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 process automation, integrations, speed, and responsiveness, Veracode helps companies get accurate and reliable results to focus their efforts on fixing, not just finding, potential vulnerabilities.

Veracode serves thousands of customers worldwide across a wide range of industries. The Veracode solution has assessed more than 53 trillion lines of code and helped companies fix more than 71 million security flaws.

Learn more at www.veracode.com,
on the Veracode blog and on Twitter.