

Contents

Welcome Letter	2
Executive Summary	3
Overall State of Software Security	6
How prevalent are application flaws?	
What proportion of flaws are fixed?	
How quickly are flaws fixed?	
Does DevSecOps drive faster fixing?	
Is security debt rising or falling?	
A Look at Application Security Testing	14
How often are applications tested?	
How regularly are applications tested?	
Not All Flaws Are Created Equal	18
What types of flaws are most common?	
How common are severe and exploitable findings?	
Does flaw prevalence differ by language?	
Not All Flaws Are Remediated Equally	26
Which flaws are fixed most often?	
Is remediation out of focus?	
How fast are flaws fixed?	
The Elusive “Average”	
Which flaws are fixed the fastest?	
Breaking Down Security Debt	37
Do priorities contribute to security debt?	
Is there a security debt-to-income ratio?	
Is fix capacity a constant?	
What is security debt comprised of?	
Regional Breakouts	49
Does software security change by region?	
Key Takeaways	52
Appendix: Methodology	55

LETTER FROM TIM JARRETT, CHRIS WYSOPAL AND CHRIS ENG

Welcome to the 10th volume of Veracode's flagship report, the State of Software Security (SOSS). This is a big milestone for the application security industry, and for us — a decade of SOSS!



Tim Jarrett
Senior Director,
Product Management



Chris Wysopal
Founder and Chief
Technology Officer



Chris Eng
Chief Research Officer

As we reflect back over the past 10 volumes, we're struck by both the enormous change and growth in our industry (and in our own company), and also what has remained the same. We've seen AppSec awareness grow in leaps and bounds since we started down this SOSS path a decade ago. When we were working on SOSS Volume 1, we spent most of our time trying to explain and advocate for application security. Today, we spend far less time talking about what AppSec is, and more time talking about how to build an effective, mature application security program.

At the same time, the core problem we are trying to solve today is not that far removed from the problem we were trying to solve 10 years ago. In State of Software Security v1, we concluded that "Most software is indeed very insecure." We could use that same statement in Volume 10. However, we are seeing some positive AppSec signs in 2019. Organizations are increasingly focused on not just finding security vulnerabilities, but fixing them, and prioritizing the flaws that put them most at risk. Though vulnerabilities are introduced as part of the development process, the data suggests that finding and fixing vulnerabilities is becoming just as much a part of the process as improving functionality.

Even with the strides the industry has made over the past 10 years, there's plenty of room for improvement — especially regarding the time it takes to make those fixes. In talking with our customers, and examining the data we used for this year's report, the notion of security debt has emerged as a significant pain point. Just as with credit card debt, if you start out with a big balance and only pay for each month's new spending, you'll never eliminate the balance. In AppSec, you have to address the new security findings while chipping away at the old. Easier said than done, but we unearthed some data points for this year's report that shed light on a path forward, and highlight some of the practices that help our customers tackle their security debt. This year's analysis highlights compelling evidence that a steady, regular scanning cadence not only improves fix rates, but also lightens the security debt load.

Thanks for being part of this big milestone on our AppSec journey. We started Veracode with a mission to secure the world's software. Today, that mission remains, with the added focus of enabling you to create, innovate, and "change the world" with software, without being held back by security concerns. We hope the best practices outlined in this report play a role in that goal.

Here's to the next 10!

SINCERELY,

Tim Jarrett, Chris Wysopal and Chris Eng



Executive Summary

In 2011, Marc Andreessen wrote an article in the *Wall Street Journal* that included the now-famous phrase “software is eating the world.” Eight years on, that statement rings truer than ever. It’s not a stretch to say that software is eating the cybersecurity world as well. The fallout from not integrating security early in the development lifecycle has never been more apparent. And our annual report on the State of Software Security (SOSS) has never been more important.

This year also marks an important milestone for the SOSS itself. It's our 10th edition! We've observed and learned a lot over the last decade producing this report, and we're especially excited to share some "*Then vs. Now*" comparisons.

Number of applications tested:

VOL. 1	1,591	VOL. 10	85,000
--------	-------	---------	--------

That's a growth of 50x!

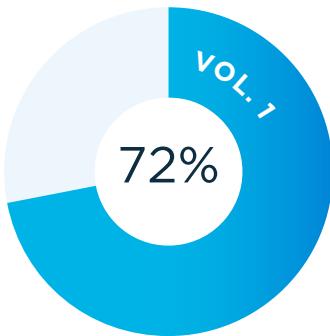


Average number of days to fix flaws:

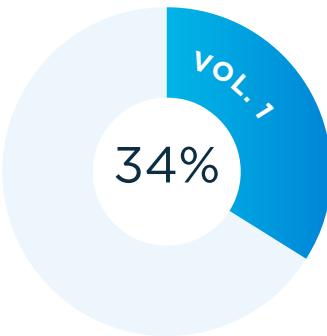
VOL. 1	VOL. 10
59	171

But the median remained 59 days. This indicates most fixes happen quickly, but there's a long and growing tail of unresolved findings.

Applications with at least one flaw:



Applications with high-severity flaws:



That's an increase of 11%!



That's a decrease of 14%!

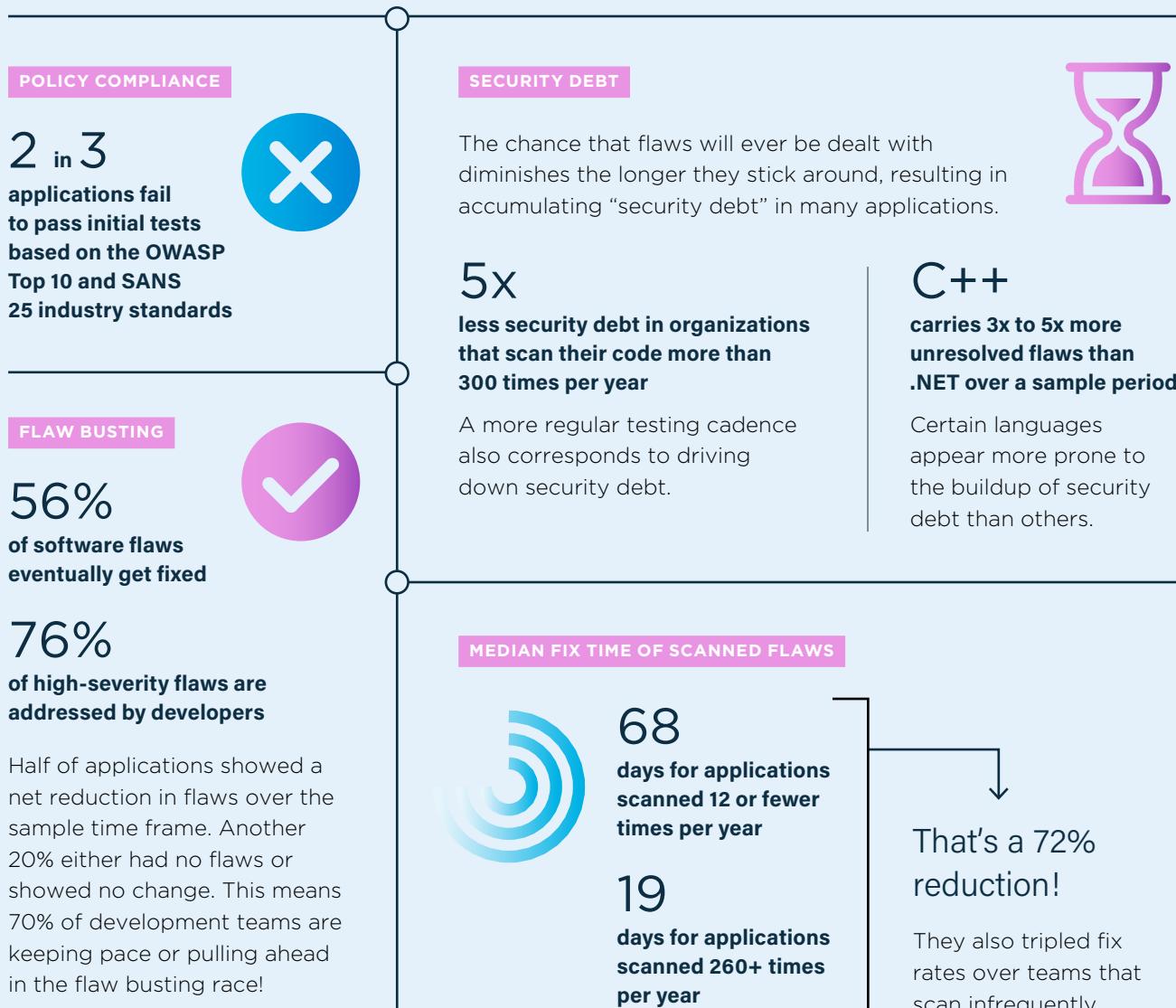
Pass rate for OWASP Top 10 policy scans:

IMPROVED BY ALMOST 10%

Beyond those 10-year views, we learned more about the state of software security in 2019.

Those who read last year's SOSS may remember a heavy emphasis on flaw persistence timeframes and what contributes to making them longer or shorter. We return to that topic this year, but focus on the accumulating security debt in applications caused by those persistent flaws and long fix timeframes.

Here are some key findings we'll expound on in this report:



CORE LESSON

It's a near certainty that your applications have security flaws of various types. The likelihood of remediating those flaws in a comprehensive and timely manner is not nearly as certain. The ability to do this consistently — and thereby driving down security debt rather than racking it up — is what separates leading and lagging SDLC programs.



Overall State of Software Security

As stated previously, this is the 10th edition of the SOSS. As we review what the data tells us about important trends over the last year, it makes sense to reflect back on what we've seen during the last decade as well. Let's do that now, in fact, starting with a data point that shows just how much the SOSS has grown over the years.

Way back in Volume 1, we studied scan results from 1,591 applications. In Volume 10, we have the privilege of testing over 85,000 applications. That's over a 50-fold increase in sample size!

That's pretty impressive, but perhaps even more so is the level of depth we're now able to achieve in that analysis. We've teamed up once again with the data scientists and storytellers at the Cyentia Institute to level up that analytical prowess to maximize value to our readers. And with a massive dataset spanning 85,000 applications, 1.4 million scans, and nearly 10 million security findings at our disposal, you're in for an analytical treat in the pages that follow!

Number of Apps Tested in SOSS Volume 1 vs. Volume 10

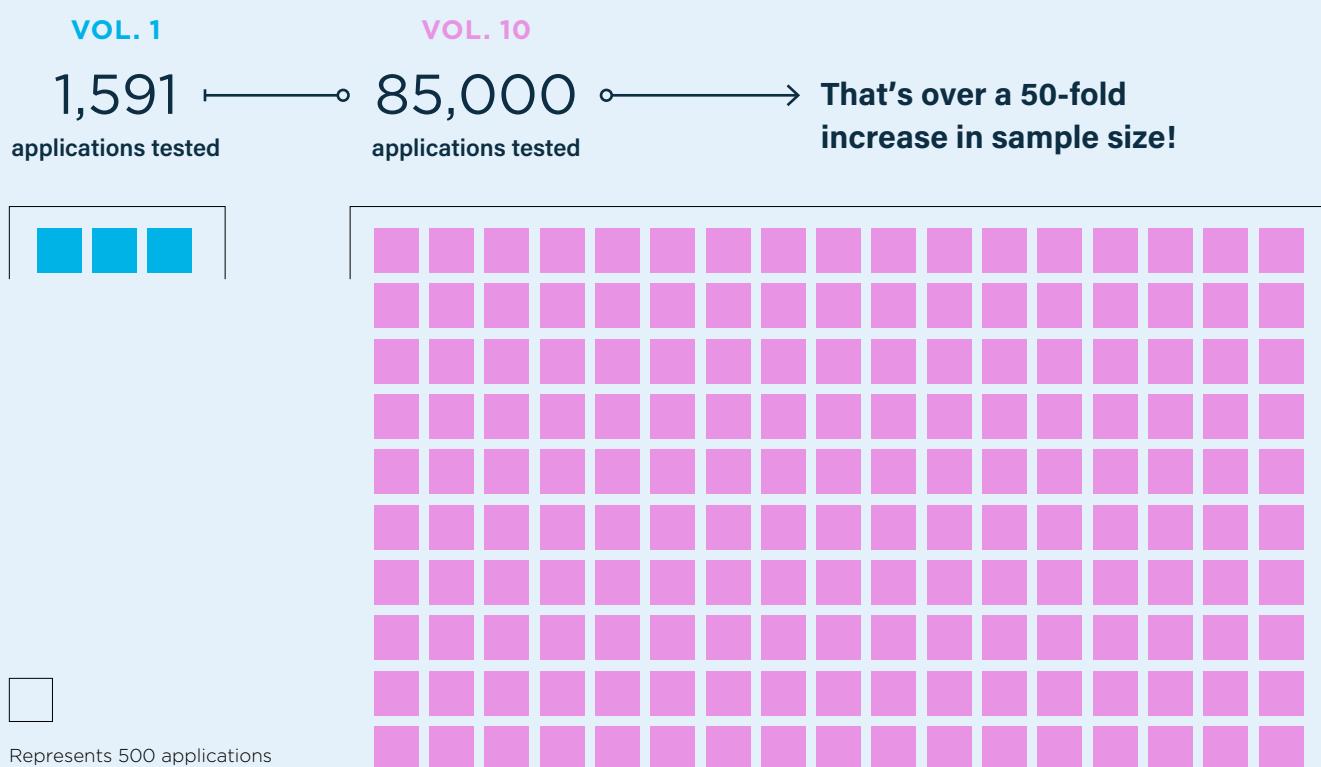


FIGURE 1 Comparison of the number of apps tested in SOSS Vol. 1 vs. Vol. 10

Source: Veracode SOSS Vol. 10



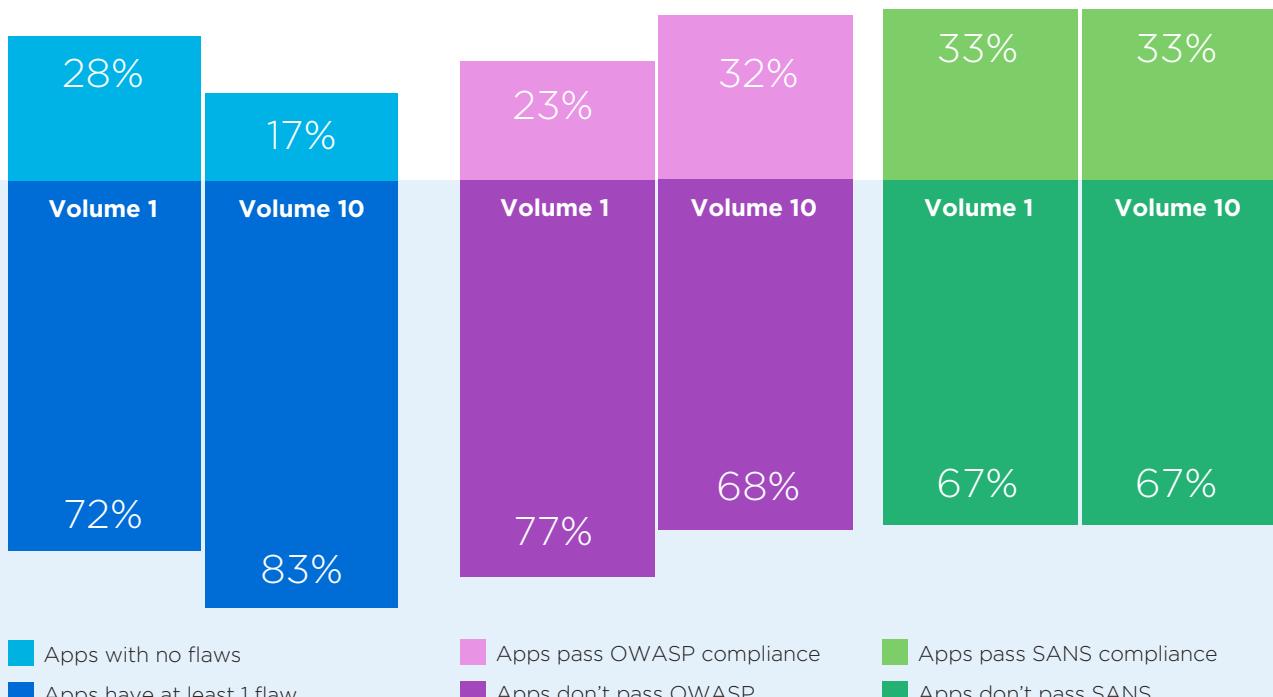
JARGON WATCH

Flaw Prevalence

The proportion of applications that have a (type of) flaw.

How prevalent are application flaws?

This first question seems simple on the surface but gets deep pretty quick. We'll dip a toe into those waters now, and wade in progressively deeper through the report. We've already mentioned that we discovered about 10 million flaws across 85,000 applications. Beyond that, 83% of those applications had at least one flaw in the initial scan run by customers. That's squarely within the range of our most recent volumes, but somewhat higher than the inaugural prevalence of 72% recorded way back in Volume 1. We attribute that upward shift to the broader set of applications tested and expanded scanning capabilities developed over that timeframe.



Beyond overall prevalence, we closely track the OWASP Top 10 vulnerabilities and SANS 25 software errors because of their status as consensus listings of the critical flaws across the industry. The pass rate for OWASP Top 10 compliance on the initial scan reversed a three-year decline by rising to 32%. That's not the highest ever recorded — that peak happened in 2016 — but the 10-year trend in Figure 3 shows things are moving in the right direction. The pass rate on tests based around the SANS 25, surprisingly, matches exactly what we tested in Volume 1.

We now know that most applications are flawed, but how serious are those findings? Overall, we discovered high-severity (level 4 or 5) vulnerabilities in 20% of applications, a 14% improvement over the equivalent statistic measured 10 years ago. Thus, the overall prevalence of findings rose over the last decade but fewer of them constitute a serious risk to applications. If you want more information on the types of flaws discovered and which ones are considered more severe, sit tight. Many pages lie ahead.

What proportion of flaws are fixed?

Prevalence conveys a key aspect of the state of application security, but more important still is whether these issues are dealt with in an effective and timely manner. Fix rate offers one way of looking at that and measures the proportion of discovered flaws that are closed or remediated. The overall fix rate across all flaws is 56%, which lands right in the neighborhood of recent years (52% in 2018; 58% in 2017).

Logic holds that not all flaws are fixed with equal urgency, and the evidence presented in Figure 5 backs that conclusion. Findings in the OWASP and SANS lists, for instance, receive slightly preferential treatment over general flaws. High-severity flaws are roughly 15% to 20% more likely to be remediated than those of lower severity. Again, none of this is terribly surprising. The main takeaway is that application teams achieve better-than-average fix rates for the flaws they prioritize. We'll talk more about what gets prioritized and why later.

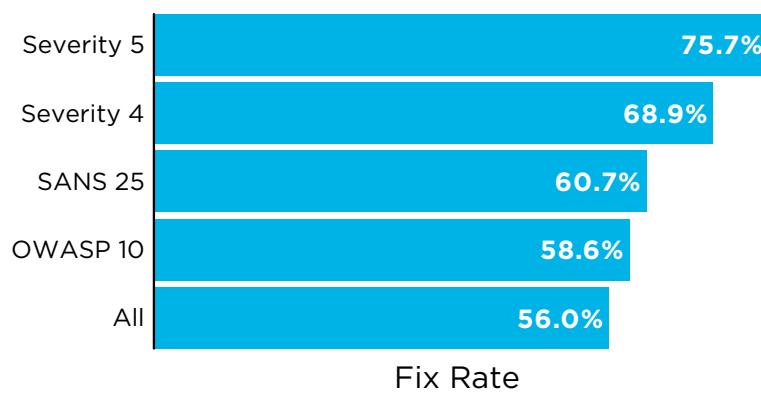
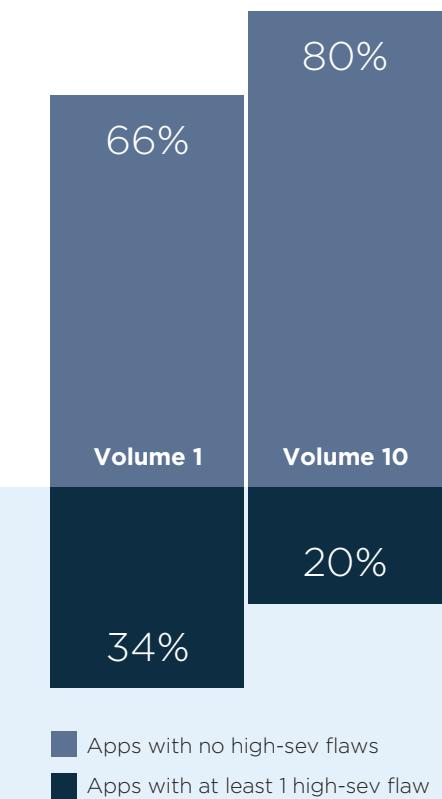


FIGURE 5 Fix rate across all flaws and for various categories of flaws

Source: Veracode SOSS Vol. 10



■ Apps with no high-sev flaws
■ Apps with at least 1 high-sev flaw

FIGURE 4

Proportion of applications with higher-severity flaws in initial scan

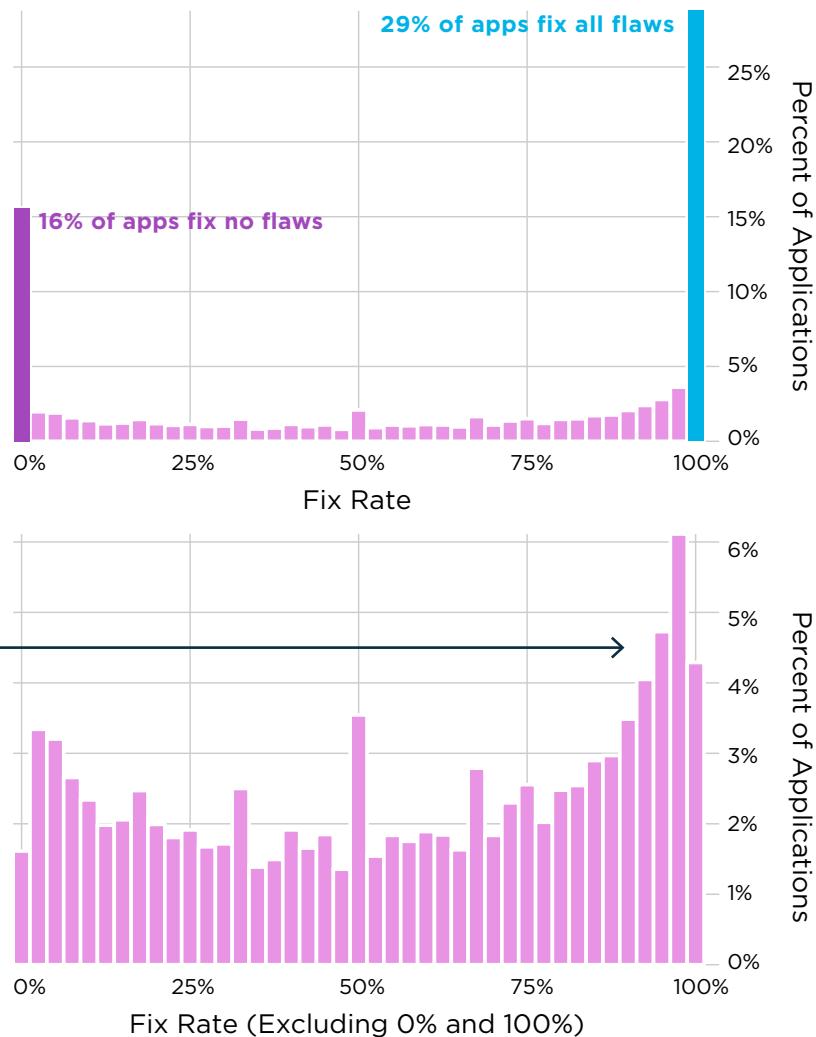
Source: Veracode SOSS Vol. 10



FIGURE 6

Distribution of flaw fix rates across applications with at least one flaw

Source: Veracode SOSS Vol. 10



Another interesting angle is how fix rate applies to individual applications. Figure 6 grants us that perspective. On the top, we see that development teams fix nothing for 16% of applications and successfully close all flaws in 29% of apps. Upon further investigation, we noted that many applications at these opposing ends of the spectrum had very few flaws.

Because these opposing extremes dominate the scale in Figure 6, we removed them from the bottom chart to focus more closely on the majority of applications that fall in the middle. Other than the spikes at 33%, 50%, and 66% that show the influence of small denominators (i.e., 1 of 3 fixed, 1 of 2, 2 of 3), there's a nice concave shape of the "bridge" connecting the two extremes.



JARGON WATCH

MTTR

The mean (average) time it takes to fix flaws discovered in an application.

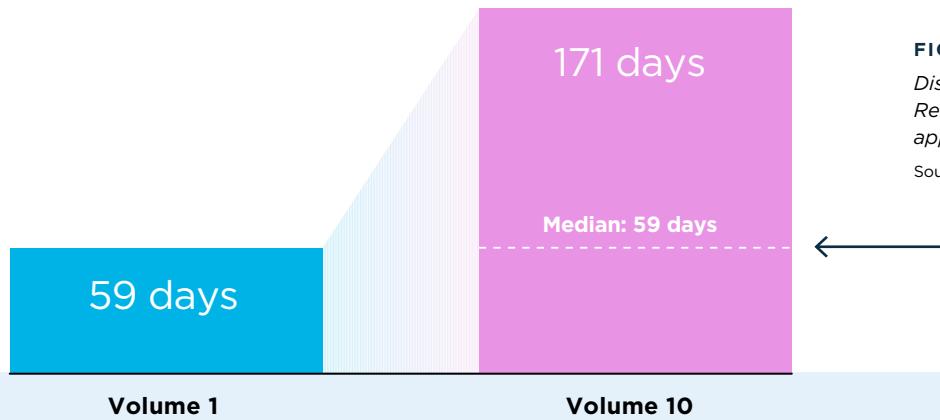
MedianTTR

The median time it takes to fix flaws discovered in an application.

How quickly are flaws fixed?

Having covered the overall fix rate for application flaws, we now turn attention to how long it takes development teams to roll out those fixes. Readers of last year's SOSS may remember a heavy emphasis on fix timelines using survival analysis techniques. We'll delve even deeper into that topic this time around; let's start simple with the common measure of Mean Time to Remediation (MTTR). As the name implies, MTTR measures the average time it takes to remediate flaws.

Figure 7 contrasts the MTTR observed in SOSS Volume 1 with that of our current sample. The results are eye-opening, to say the least. MTTR nearly tripled over the ensuing decade, raising the question of what's going on with software security in the 2010s. That comparison is deceptive, however, because the average of flaws suffers from the flaw of averages.



Not to be overly mean,¹ but the average becomes an unreliable measure of "typical" values in skewed distributions. And time-to-remediation creates a very long-tailed distribution (take a sneak peek at Figure 21 if you want proof). That long tail is comprised of unresolved findings that inflate the MTTR. By way of comparison, the median time-to-remediation (MedianTTR) of flaws in the development cycle during the last year is just two months — equal to the MTTR from back in the SOSS Volume 1 report. Thus, typical fix times haven't gotten worse; the tail of ever-accruing "security debt" just got a lot longer.

FIGURE 7

Distribution of Mean Time to Remediation among closed application flaws

Source: Veracode SOSS Vol. 10

Figure 7 shows a tripling of average fix time over the last decade, which seems to suggest software security may have lost its way. But the median fix time remains unchanged from 10 years ago. Thus, typical fix times haven't gotten worse; the tail of ever-accruing "security debt" just got a lot longer.

¹ Stats jokes always regress to being mean.

Does DevSecOps drive faster fixing?

As with financial debt, escaping out from under security debt necessarily requires changing habits to pay down balances. The integration of software development and IT operations (DevOps) and integration of security into those processes (often called DevSecOps) over the last several years has certainly changed habits. We do not have a definitive way to distinguish development teams that practice Dev(Sec)Ops, but we can look for certain observables tied to behaviors in keeping with that spirit.

The frequency and cadence of security testing are two such observables. In general, we expect a DevOps-oriented team to conduct frequent security scans of their code at regular intervals during the development lifecycle. Furthermore, we'd hope to see evidence that those behaviors correlate with faster fix timelines.

Figure 8 shows that hope has some merit. The MedianTTR for applications scanned 12 or fewer times a year (less than once per month, on average) stands at 68 days. Those with an average scan frequency of daily or more (260+ scans²) knocked that statistic way down to 19 days. That's a 72% reduction in MedianTTR.

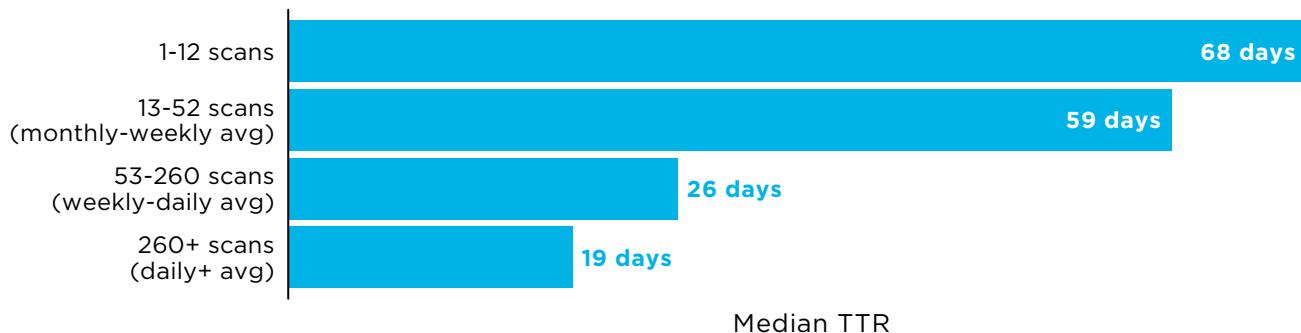


FIGURE 8 Effect of scan frequency on fix rate and time-to-remediation

Source: Veracode SOSS Vol. 10

It's not shown in Figure 8, but those same frequent scanners also tripled their fix rate and reduced security debt by five-fold! "I should scan my apps more often" is the smart mental note to make here.



JARGON WATCH

Fix Capacity

The number of flaws a development team can close relative to the number of flaws discovered. Usually expressed as a negative or positive ratio.

Is security debt rising or falling?

That notion of security debt brings us to arguably the most defining indicator of the state of software security in 2019 — whether applications are accruing or eliminating flaws over time. To that end, Figure 9 measures the overall fix capacity of development teams by comparing the number of flaws found in an application's first and last scans.

Overall, 30% of applications show an increased number of flaws in their latest scan. This doesn't necessarily imply those teams are doing a bad job managing flaws — it could represent a period of rapid growth and change — but it does reveal evidence of accruing security debt. If these applications are on a path similar to those of virtuous venture-backed startups, then we hope to see them escape their negative security burn rate in the near future.

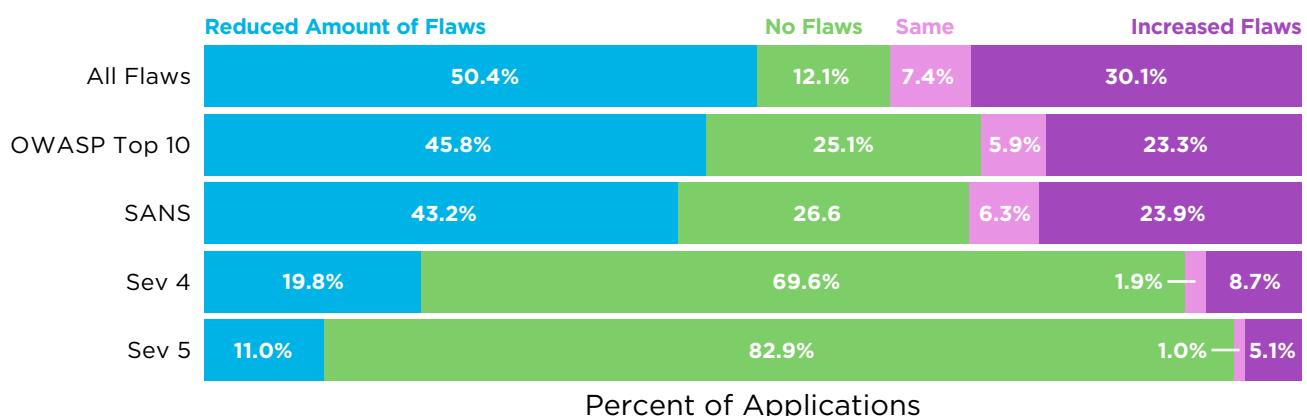


FIGURE 9 Difference in the number of flaws found between first and last scans of sample period

Source: Veracode SOSS Vol. 10

Half of application teams drove down flaws over the sample time frame. Another 20% either had no flaws (12%) or showed no change (7%). This means a respectable 70% of development teams are keeping pace or pulling ahead in the flaw busting race! What's more, that win record jumps to over 90% for high-severity flaws. We view this as a positive sign that the overall state of software security is on a positive trajectory in 2019. But there's still a lot of work to be done and many lessons to be learned, and supporting those endeavors is exactly what the rest of this report is all about. Let's dive in.



A Look at Application Security Testing

Given that this is a report about software security, it makes sense to review how organizations incorporate security testing into their development processes. We walk through a few brief Q&As in this section to help level set for the analysis of testing results in the sections that follow.



JARGON WATCH

Scan Frequency

The number of application security scans conducted over a period of time.

How often are applications tested?

This is a tricky question because organizations test many different types and sizes of applications in various stages of development. Nevertheless, there's still value in an overall scan rate across all applications because it provides an indicator of security interest and activity among development teams. Readers of last year's SOSS Volume 9 may remember that more frequent scanning correlated with a marked improvement in remediation timeframes. In that light, Figure 10 contains both encouraging and discouraging signs.

A little north of one in three applications received just a single scan during the year. Granted, some of those applications represent software introduced late in the sample period or code that died on the vine or was grafted into a larger application. But many of them are legitimate applications that would almost certainly benefit from more security attention than they're currently getting.

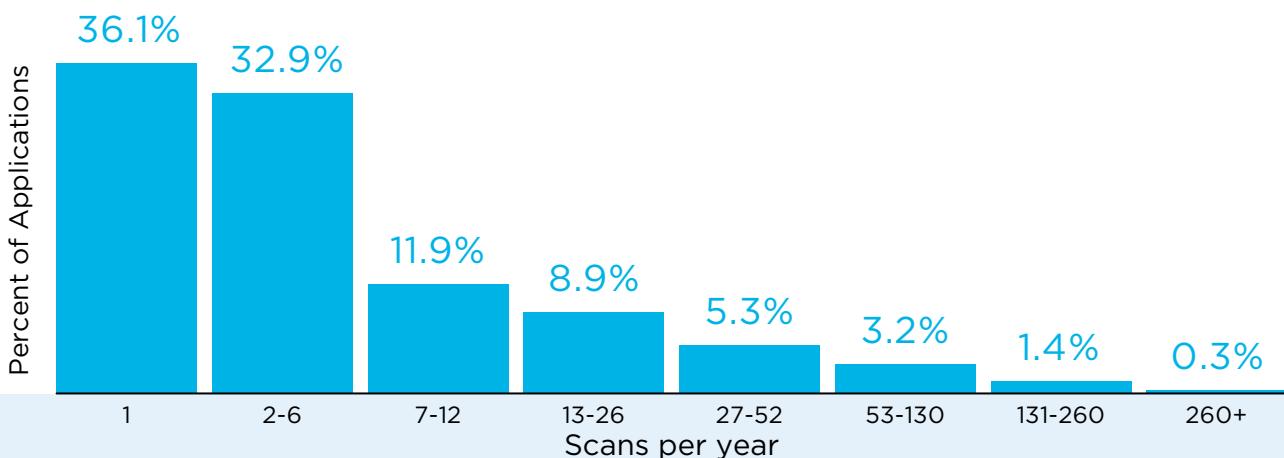


FIGURE 10 Frequency of security scanning across applications

Source: Veracode SOSS Vol. 10

After those one-and-done scanners, we see another third of applications received two to six scans and another 12% recorded seven to 12. Adding those up determines that 80% of applications average one scan per month or less. Note, however, there's no guarantee those scans occurred regularly on monthly intervals. We checked into that, in fact, and uncovered some interesting observations about scanning cadence that we hit in the next section.



JARGON WATCH

Scan Cadence

A measure of the regularity of application security scans over a period of time.

Before we go there, the 20% of applications scanned more than 12 times in the year warrant special mention (and kudos). As previously mentioned, these frequent scanners are significantly faster than the average bear at busting flaws. They lean slightly toward larger, more business-critical applications coded in enterprise software languages like .NET and Java (COBOL and VB6 exhibit the lowest scan rates). Beyond that, though, we see no consistent defining characteristics among these applications and are left to assume that scanning frequency depends more on the coders than the code. Clearly the integration of security into continuous development practices has a way to go to be truly universal.

How regularly are applications tested?

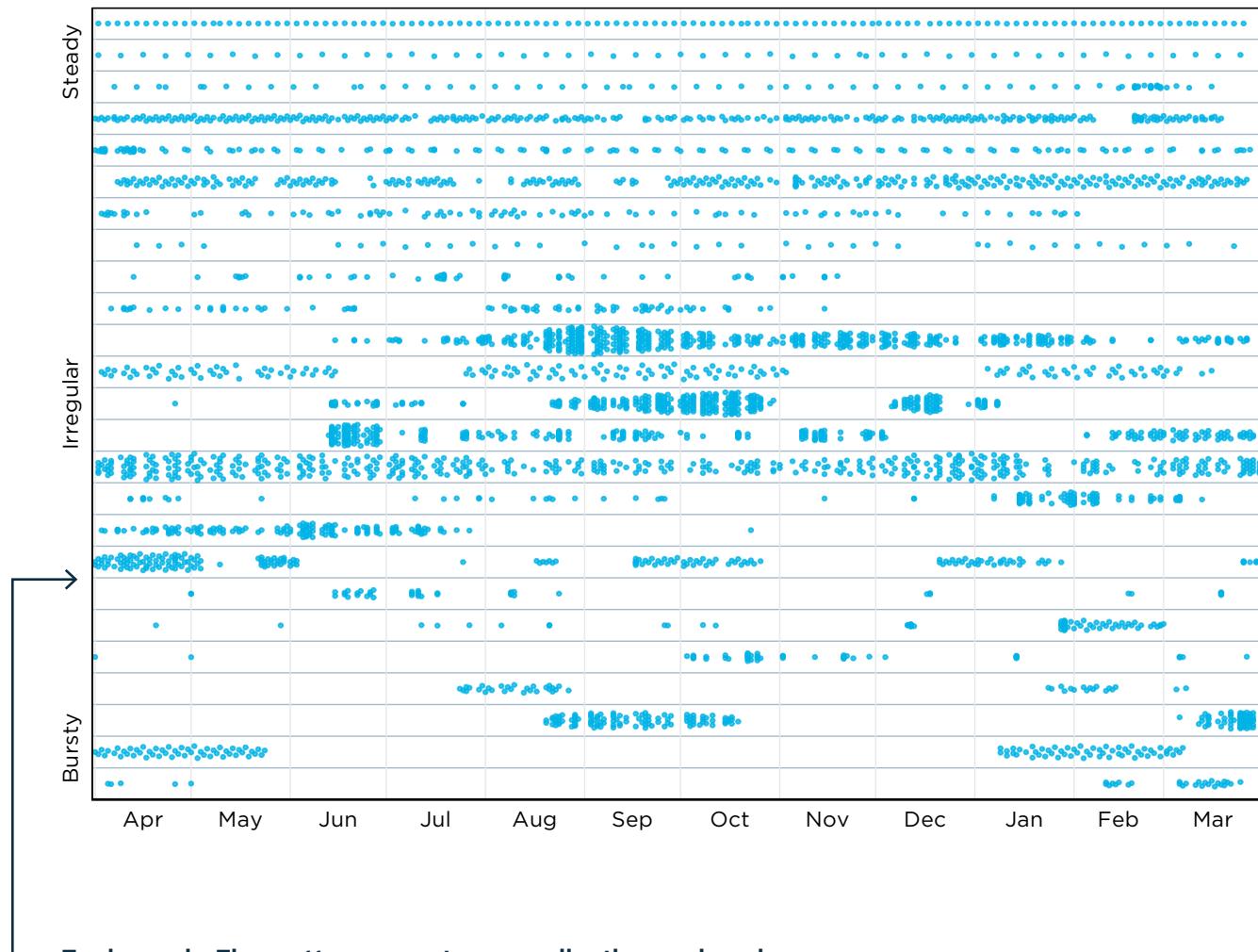
In addition to the frequency of application scanning, we can measure the cadence of scans over time. Some development teams conduct tests at very regular intervals, while others take a more irregular, or bursty approach. This regularity (or lack thereof) is a challenge to capture in a single number. To measure scan cadence we employ something called the Fano factor, which is simply the ratio between the variance of time between scans and the average time between scans. Regular scanning means low variance and a low Fano factor, waiting and then repeatedly scanning in a short period means a high variance and low mean, ergo a high Fano factor.

Figure 11 shows what this looks like in practice over the course of one year. Each row represents an application and each dot represents a scan of said application. Applications at the top exhibit a steady cadence, those on the bottom are the most bursty, and the middle are classified as irregular.

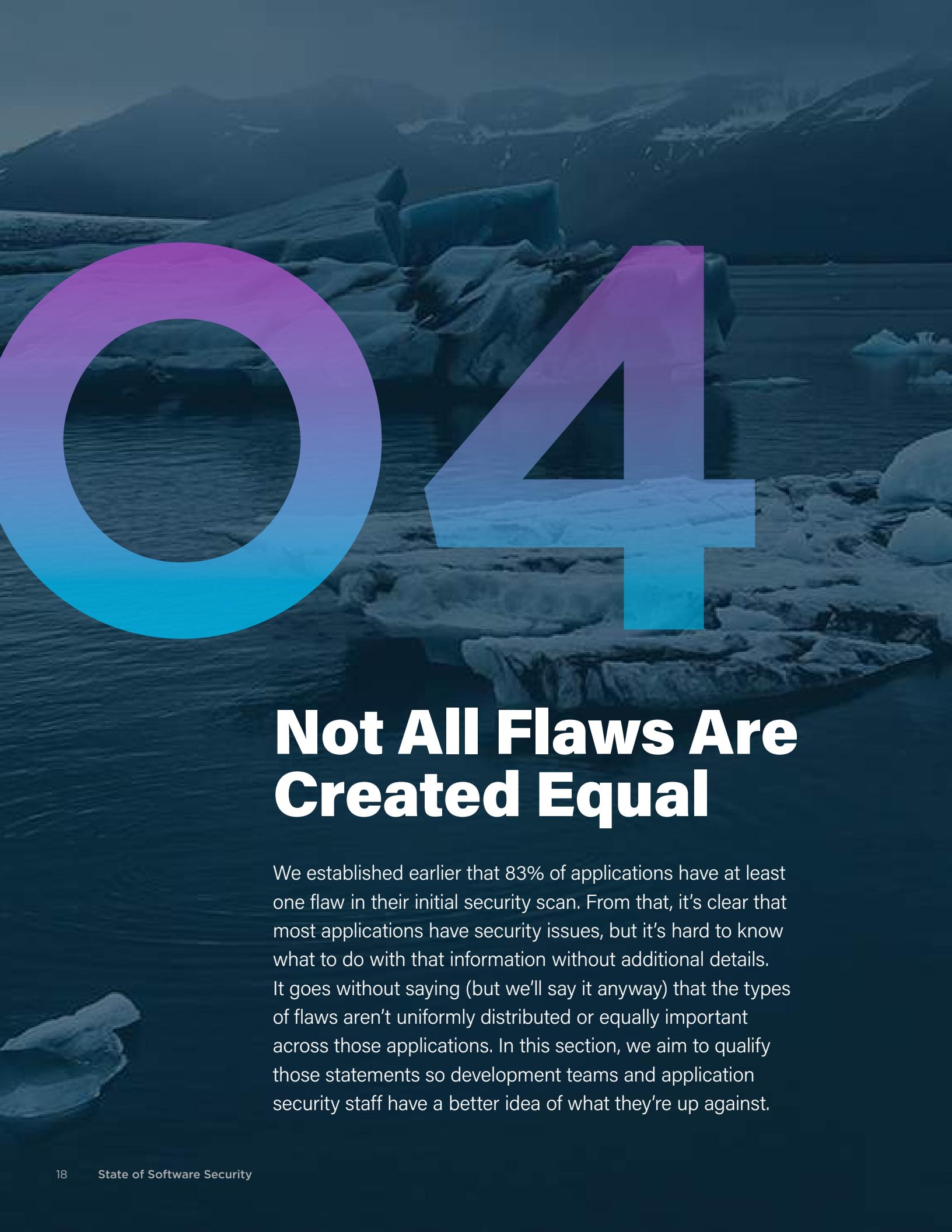
Similar to frequency, there's no magic cadence that separates good and bad practice. Bursty scanning could be in keeping with a waterfall development cycle or an event-driven testing schedule. Regular scanning could indicate a DevSecOps orientation, but it may simply reflect ordinary scheduled scanning. We'll return to the topic of scanning cadence and frequency a bit later and test whether either correlate with the reduction of security debt.

FIGURE 11
Cadence of security scanning across a sample of applications

Source: Veracode SOSS Vol. 10



Each row in Figure 11 represents an application and each dot marks a security scan of said application. Applications at the top exhibit a steady cadence, those on the bottom are the most bursty, and the middle can be classified as irregular.



Not All Flaws Are Created Equal

We established earlier that 83% of applications have at least one flaw in their initial security scan. From that, it's clear that most applications have security issues, but it's hard to know what to do with that information without additional details. It goes without saying (but we'll say it anyway) that the types of flaws aren't uniformly distributed or equally important across those applications. In this section, we aim to qualify those statements so development teams and application security staff have a better idea of what they're up against.

What types of flaws are most common?

Let's begin answering this question by stepping into the wayback machine to retrieve some stats from SOSS Volume 1 on the most common flaws. Updating that with our most recent results in the 'Now' column creates the nifty 10-year trajectory across flaw categories found in Figure 12.

The top two flaw types from Volume 1, Cryptographic Issues and Information Leakage, remain the same 10 volumes later but swapped places. CRLF and Insufficient Input Validation appear to be the top two gainers over the decade, followed closely by Credential Management. This may be due more to broadening scanner coverage than any major prevailing trend.

Buffer Overflow, Buffer Management Errors, and Numeric Errors weigh in as the decade's biggest losers. This is consistent with the decline we noted earlier of C++ as a coding platform. With the rise of JavaScript and .NET, buffer management is more often handled by the language itself, reducing the prevalence of related flaw categories.

The top two flaw types:

VOL. 1

1. Cryptographic Issues
2. Information Leakage

VOL. 10

1. Information Leakage
2. Cryptographic Issues

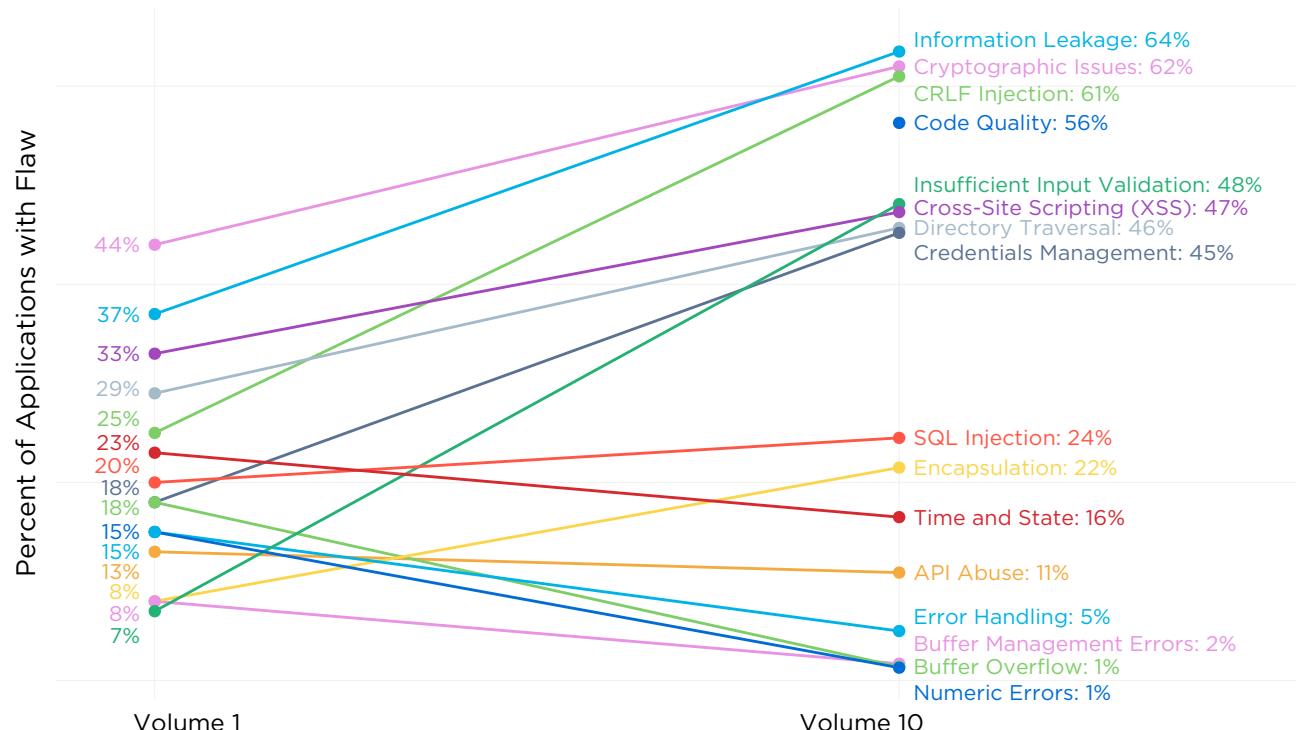


FIGURE 12 Prevalence of flaw categories in SOSS Volume 1 and 10

Source: Veracode SOSS Vol. 10



JARGON WATCH

Flaw Intensity

The number of flaws present per application (when discovered).

Following that little jaunt down memory lane, we now take a closer look at present-day application security findings. Figure 13 presents flaw categories detected by static analysis (SAST) along two dimensions: prevalence and intensity. Prevalence measures the proportion of applications that exhibit a given type of flaw, while intensity captures the volume of those flaws when detected. Both tell us something about frequency but from different perspectives.

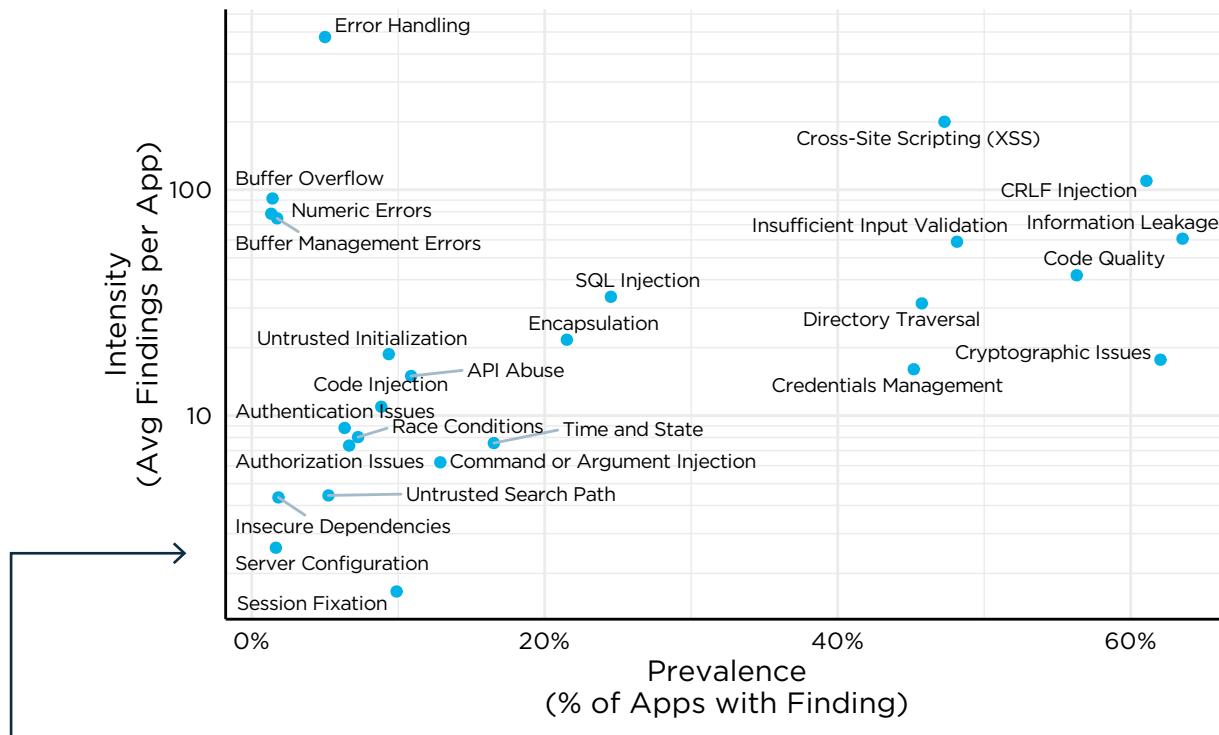


Figure 13 (and several following) presents the commonality of flaw categories on two dimensions: prevalence and intensity. Prevalence measures the proportion of applications that exhibit a given type of flaw, while intensity captures the volume of those flaws when detected.

FIGURE 13 Prevalence and intensity of flaw categories discovered by static analysis
Source: Veracode SOSS Vol. 10

There's a lot to take in from Figure 13, so we suggest consuming it by quadrants. Starting in the top left quadrant, we find flaws that are relatively rare overall (low prevalence) but tend to show up in droves (high intensity). These often represent endemic issues for particular types of applications, languages, etc. Error Handling flaws, for example, are often found among C/C++ applications as well as those based on IBM's old RPG 4 language.

The top right quadrant features flaws that affect numerous applications in great numbers. Thus, XSS and CRLF Injection can be considered pandemic flaws across the application landscape. That doesn't necessarily mean they represent the gravest risk, but it does imply an incessant plague on development programs that carries a high cost or consequence or both.

The sparsely populated lower-right quadrant represents widespread “point” issues. Specifically, flaws falling under Credentials Management and Cryptographic Issues are common, but fortunately they’re not typically found in many unique areas of an application.

The bottom left is rather crowded, but that’s actually a good thing. Anything listed there is rare with a low per-app intensity. May all your flaws move in that direction.

We apply the same visual technique to flaws discovered via dynamic analysis (DAST) in Figure 14. This noticeably rearranges the distribution of flaws across the grid and highlights the different (yet complementary) capabilities offered by SAST and DAST. The much higher prevalence of potential Server Configuration issues exemplifies this; such vulnerabilities have more to do with the environment in which applications run than the codebase itself. We’ll let you explore other comparisons as you wish. The main point is that both dynamic and static scans have a place in application security and both have a story to tell in our data.

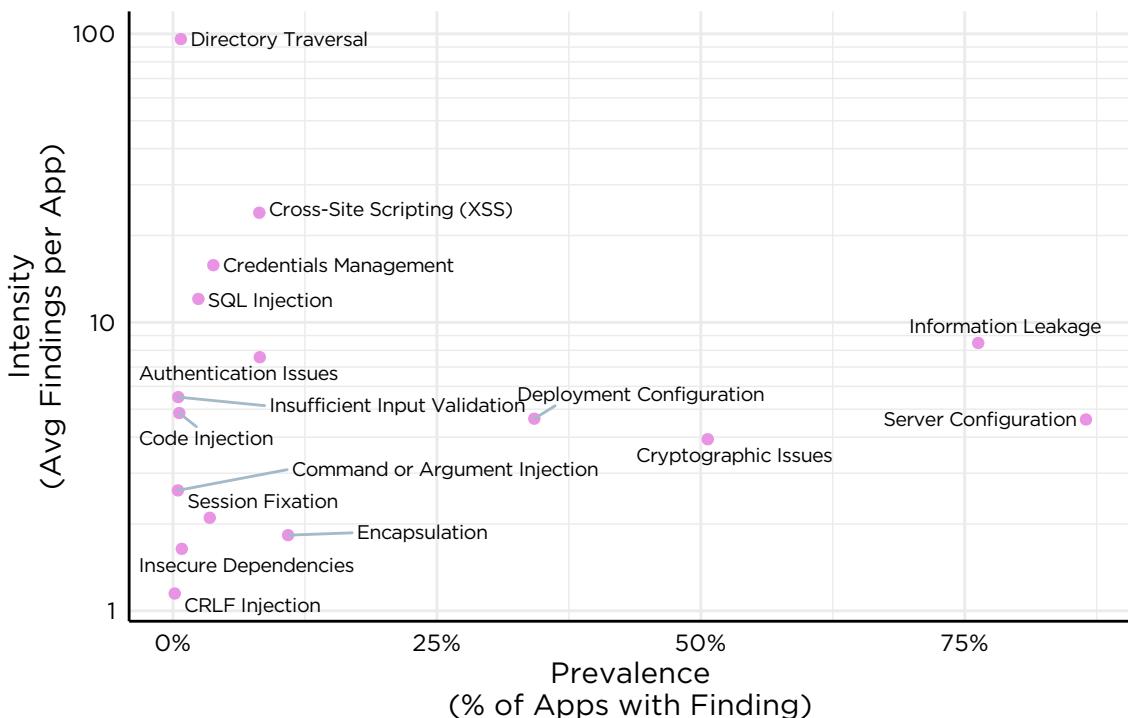


FIGURE 14 Prevalence and intensity of flaw categories discovered by dynamic analysis

Source: Veracode SOSS Vol. 10

How common are severe and exploitable findings?

Looking over the previous charts, you may have thought something to the effect of “yeah, but some flaws are worse than others.” And you’d be onto something. We assess the severity and exploitability of all flaws so that application security teams can better prioritize their remediation efforts.

Severity scores reflect the potential impact of any given flaw to the confidentiality, integrity, and availability of the application. In general, higher severity flaws are less complicated to attack, more prone to remote exploitation, and allow full application compromise. A fuller description of these scores can be found in the table below for reference.

Severity Score	Severity Level	Description
5	Very High	The offending line or lines of code is a very serious weakness and is an easy target for an attacker. The code should be modified immediately to avoid potential attacks.
4	High	The offending line or lines of code have significant weakness and the code should be modified immediately to avoid potential attacks.
3	Medium	A weakness of average severity. These flaws should be fixed in high assurance software. You should consider fixing this weakness after you fix the very high and high flaws for medium-assurance software.
2	Low	This is a low priority weakness that will have a small impact on the security of the software. You should consider fixing these flaws for high-assurance software. Medium- and low-assurance software can ignore these flaws.
1	Very Low	Minor problems that some high-assurance software may want to be aware of. These flaws can be safely ignored in medium- and low-assurance software. This year’s data found these flaws only in manual and dynamic scans — static data analyzed in this section does not include flaws in this severity level.
0	Informational	Issues that have no impact on the security quality of the application but which may be of interest to the reviewer.

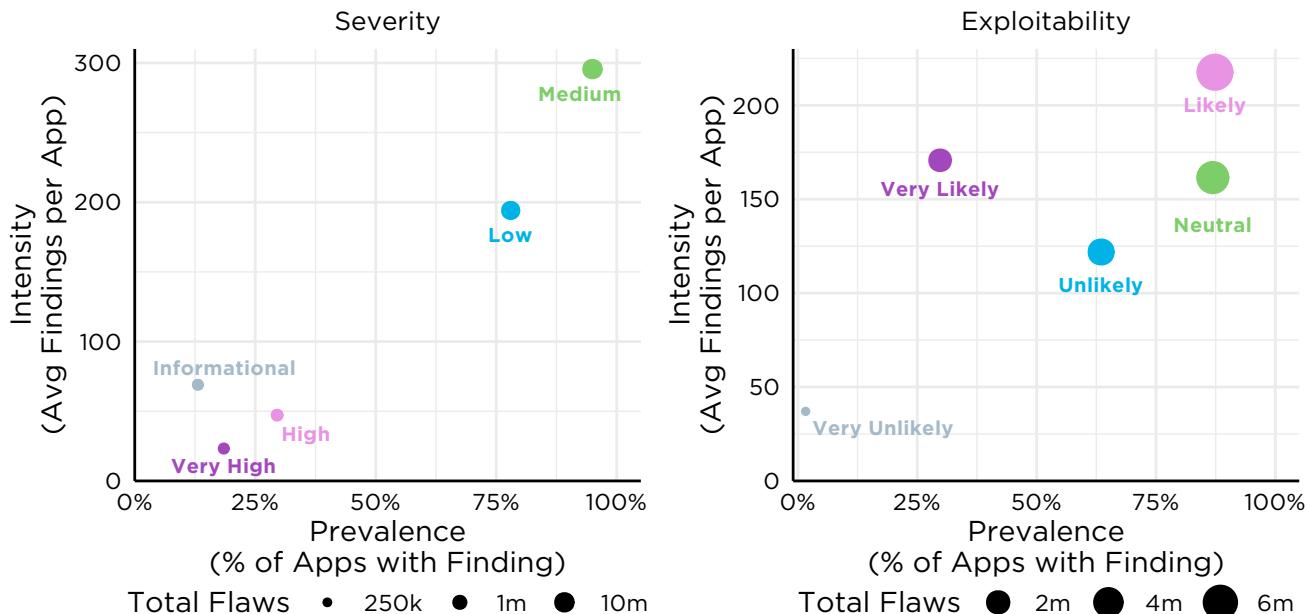


FIGURE 15

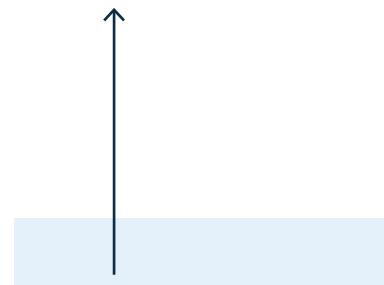
Prevalence and intensity of flaws categorized by severity and exploitability

Source: Veracode SOSS Vol. 10

The leftmost chart in Figure 15 plots the prevalence, intensity, and volume (dot size) of flaws within each severity level. The results generally reveal what we know and expect. High-severity vulnerabilities (Severity 4 and 5) are relatively rare across and within applications, as are those on the bottom end of the severity spectrum. The mid-range flaws clearly dominate scan results, with most applications exhibiting high frequency of severity 2 and severity 3 findings.

Exploitability adds another dimension to evaluating the importance of application security findings. While severity scores assess a flaw's overall potential impact to the application, exploitability estimates its susceptibility to attack. We measure exploitability on a scale from -2 (very unlikely) to 2 (very likely).

Flaws are aggregated by exploitability rating and depicted on the right side of Figure 15. Here we see a different pattern than for severity scores. Flaws deemed likely candidates for exploitation win the Triple Crown by leading the field according to prevalence, intensity, and volume. That's unfortunate, but a good reminder of the importance of identifying and remediating flaws during the development lifecycle. We'll see how teams are going about that a bit later.



Flaws deemed likely candidates for exploitation win the Triple Crown by leading the field according to prevalence, intensity, and volume.

Does flaw prevalence differ by language?

With the rather large caveat that some applications are mixed-language, we compare flaw prevalence among the most common languages in Figure 16. The results leave little doubt that some are more susceptible to flaws than others. Python and JavaScript boast the lowest levels, though a majority of applications coded in those languages still exhibit findings of various types. On the other end, over 90% of applications based on Android, PHP, and iOS Bitcode contain flaws.

FIGURE 16

Comparison of flaw prevalence by top application languages

Source: Veracode SOSS Vol. 10

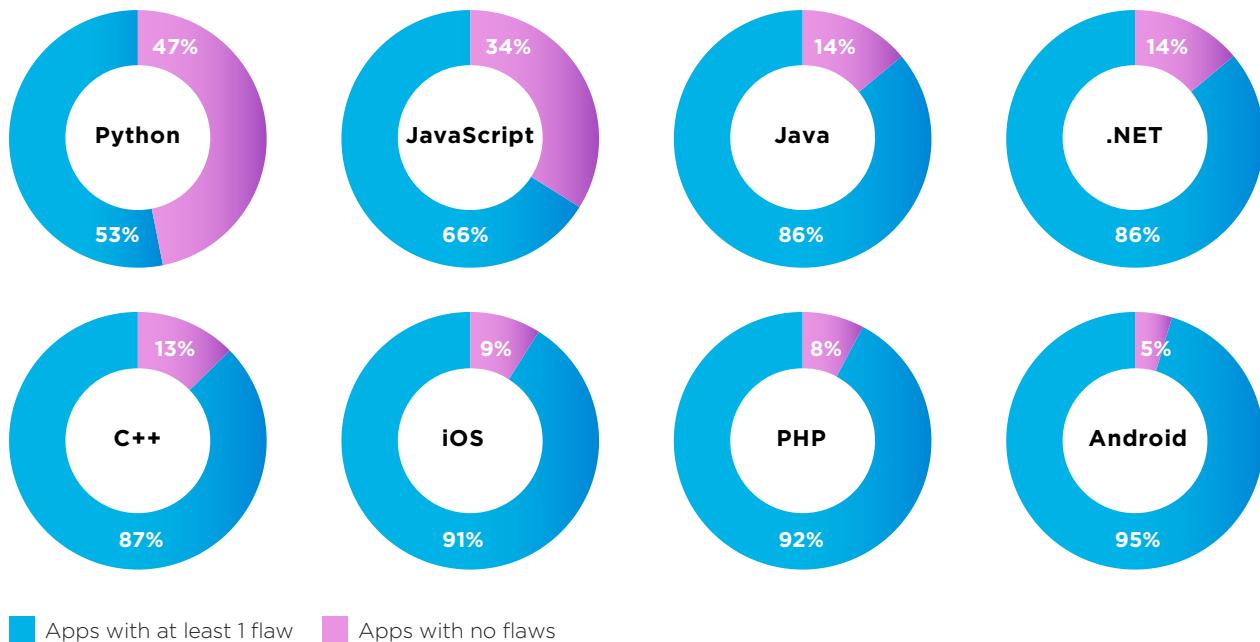


Figure 17 expands on this language-centric view of software security by contrasting the most common flaw categories on the prevalence-intensity grid we've used throughout this report. It contains a ton of detail, much of which will not be of interest unless you're responsible for applications based on certain languages. Because of that, we will just set out the buffet and let your eyes feast on whatever seems most relevant. Bon appetit!

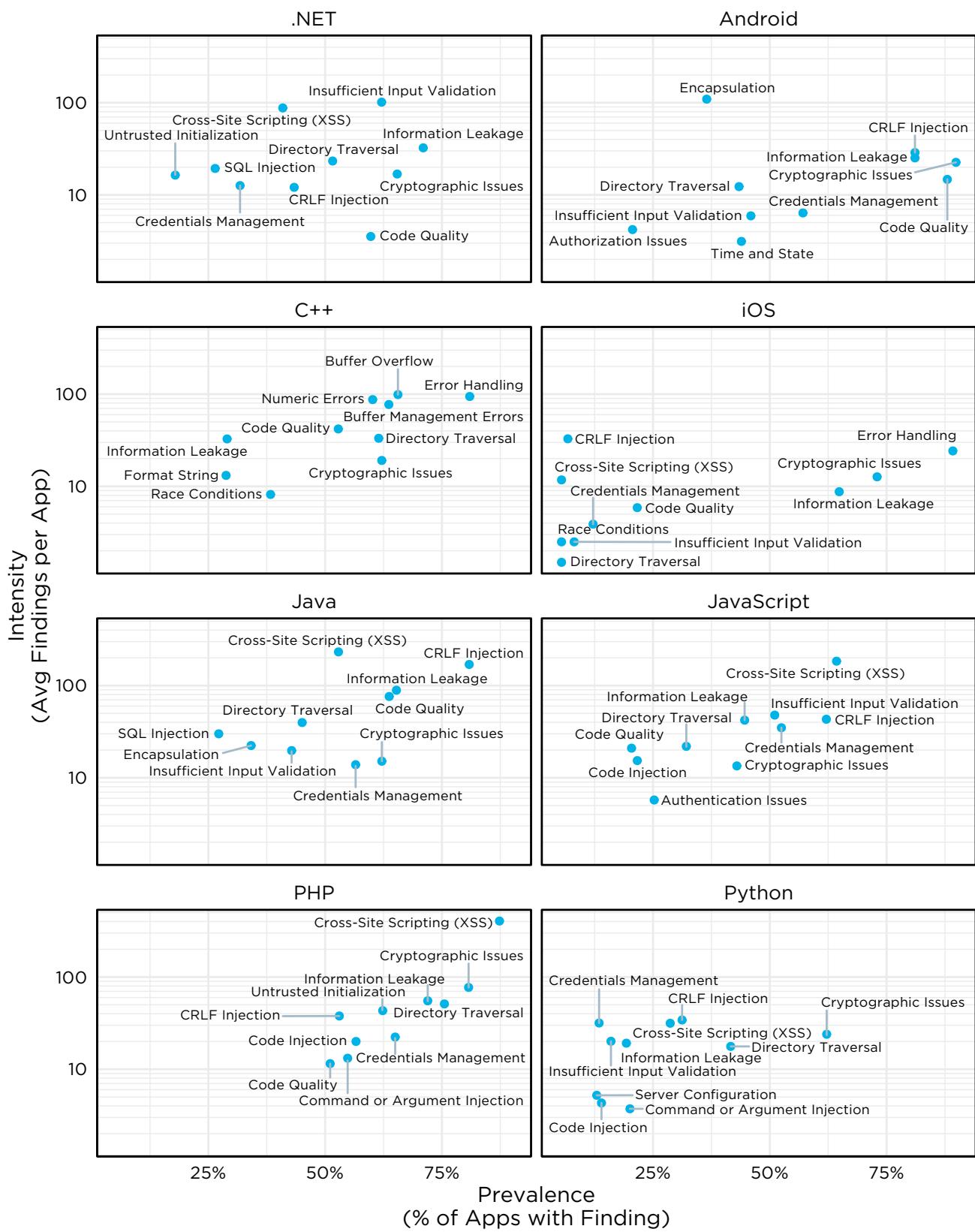


FIGURE 17 Prevalence and intensity of flaw categories by top application languages

Source: Veracode SOSS Vol. 10



Not All Flaws Are Remediated Equally

By now, it's clear that most software has flaws of one kind or another. It's the inevitable legacy of our flawed wetware. Understanding that flaws are bound to happen, the test is how teams address those issues when they inevitably surface in the development process. We assess this key aspect of software security from multiple perspectives in this section.

Which flaws are fixed most often?

When considering why certain issues get addressed while others don't, several feasible possibilities arise. Some findings are low-hanging fruit and easy to pick off. Some may be viewed as more risky. Others just happen to be next on the task list. We could go on. The point is that we cannot determine precisely why, but we can make observations about what and when from the data. The next few figures focus on the types of flaws most likely to be fixed.

Figure 18 starts things off with a trio of charts portraying fix rates across the dimensions of severity, exploitability, and application criticality. The first in that series offers good evidence that severity scores influence which findings receive attention. The likelihood of remediation increases steadily from 'Low' to 'Very High' ratings. Informational findings buck that trend, but many of those are easily addressed or just accepted for what they are — an FYI.

The contribution of exploitability and criticality scores on fix rates is less apparent in Figure 18. Flaws considered least likely to be exploited are the mostly likely to be fixed. That seems strange at first, but keep in mind that not all "fixes" involve code-level changes. Simply accepting/closing a finding works too, and that action may be warranted when the chance of exploitation is negligible and so many other things demand attention.

The pattern of fix rates by application criticality levels is curious. It doesn't seem random, yet exhibits two distinct trends. One theory that fits the data is that attention on fixing flaws increases to the point where the application's criticality to the business triggers more and more restrictive change control. Another possibility is that the criticality ratings assigned by users of the platform don't actually map to business priorities.

Fix rates for flaws contained in the SANS 25 (61%) and OWASP Top 10 (59%) lists show a minor uptick from the overall rate of 56%. There's not much to say or conclude beyond that except it appears those standards may have some degree of influence over which flaws get fixed.

We examined fix rates across flaw categories but found no obvious universal rules of remediation. Flaws with the highest likelihood of closure were not the same as those found most often among scan results. Thus, we must conclude that developers put more weight on attributes other than just prevalence when deciding which flaws should be fixed.

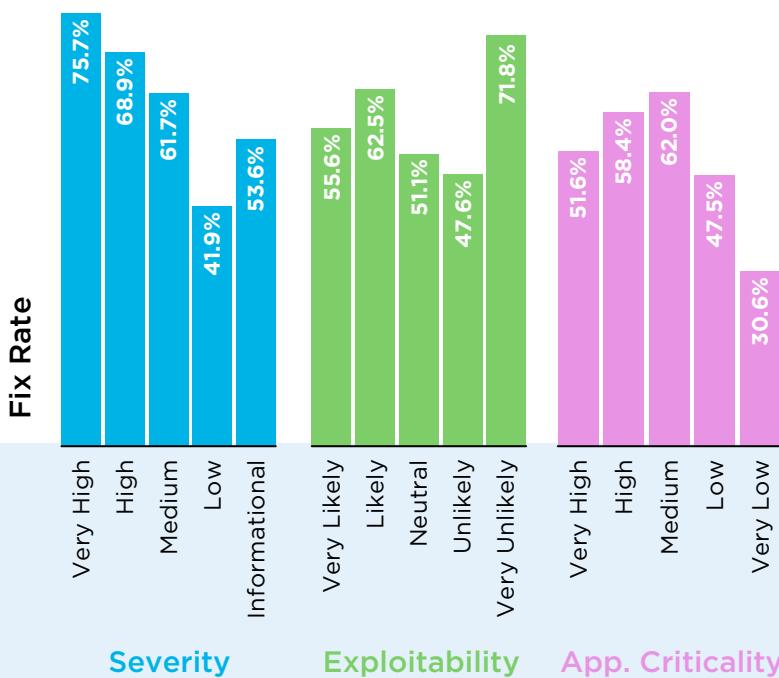


FIGURE 18

Fix rate across flaw severity, exploitability, and application criticality levels

Source: Veracode SOSS Vol. 10

Is remediation out of focus?

The analysis in this section casts a spotlight on the focus of flaw remediation efforts. Are developers prioritizing the most important vulnerabilities? How do we best define what “most important” even means? Does that meaning change within the context of each application/team/organization? These are difficult questions without clear answers but we can, at least, see if the data contains some helpful hints.

The OWASP Top 10 purposes to spread awareness of the most critical security risks to web applications.³ In that sense, it represents a broad consensus of what’s “most important.” Figure 19 shows fix rate and prevalence statistics for flaws in the OWASP Top 10. The reasons for doing this will become clear in a moment. Once again we see the pattern that the most common flaws aren’t necessarily seen as the most important (or easiest) to fix.

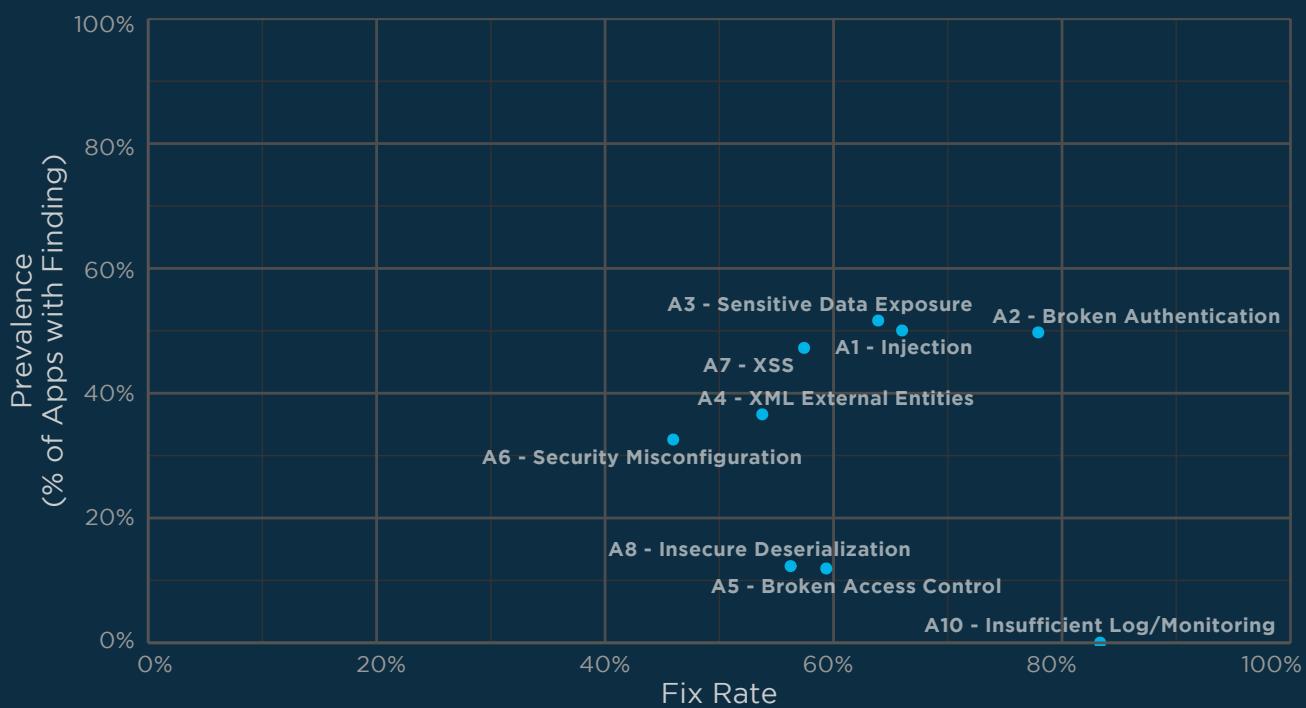
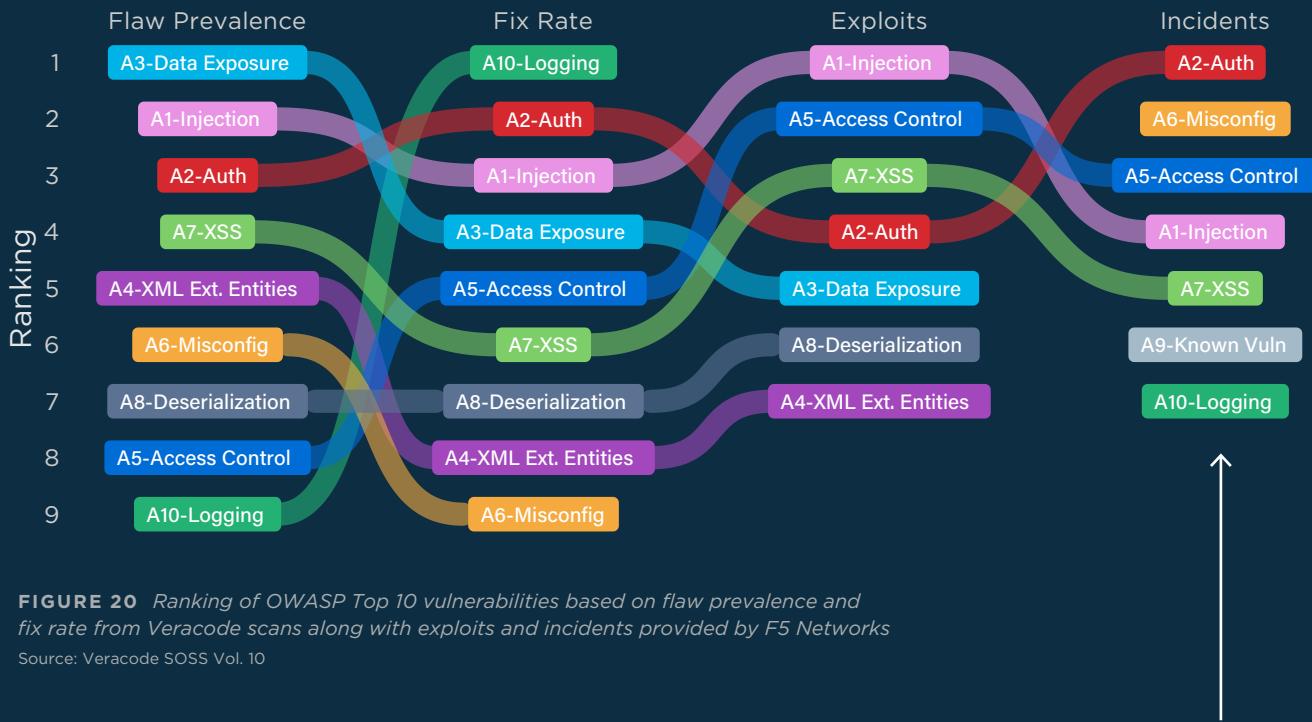


FIGURE 19

Fix statistics for flaws in the OWASP Top 10
Source: Veracode SOSS Vol. 10

Another interpretation of “most important” would include flaws that are most often used in known exploits or security incidents. We do not have access to such data, but F5 Networks does and they were kind enough to share some with us to support this analysis.



What we want to explore now is what the evidence suggests about the focus of remediation efforts in light of these two sources for determining what's "most important." In the first two columns of Figure 20 you'll find a ranking of flaw prevalence and fix rate based on Figure 19. On the right, you'll find two columns based on the data provided by F5 Networks. The first ranks OWASP vulnerabilities according to their presence in exploits recorded in Exploit Database. The second draws from a sample of incidents analyzed by their team and traced back to the root issue.

The contrast depicted across the columns in Figure 20 is fascinating. A10-Logging ranks lowest in terms of prevalence, but highest in terms of fix rate. A5-Access Control appears low in the prevalence column, yet filters toward the top of the exploits and incidents rankings. A1-Injection and A2-Authentication float around the top half across the board and A8-Deserialization consistently hangs out toward the bottom. Note that A9-Using Components with Known Vulnerabilities does not appear because such flaws do not lend themselves to detection through static analysis. Do not let that omission lull you into a false sense of complacency.

In conducting this analysis, we do not intend to imply that perfect alignment across these columns represents the ideal state. We don't have enough data to make that bold assertion. But if remediation priorities and efforts were broadly out of focus, the results depicted in Figure 20 would not cause us to question that reality.

Figure 20 asks "Are the most prevalent flaws the same as those with the highest fix rate? Are they also the ones most likely to be involved in known exploits or security incidents?" The lines show how the ranking of flaw categories shifts across those columns.

How fast are flaws fixed?

In the opening section, we pegged the average time to remediate (MTTR) flaws across our (very large and diverse) sample at 171 days and the median at 59 days, but hinted those numbers tell only part of the story. We now pick up where we left off, thicken the plot, and hopefully bring things to a satisfying resolution. Our story begins with what it means to be average.

Figure 21 illustrates the long tail inherent to flaw remediation timelines. It's meant to show the difficulty of isolating a central or "typical" measure in such a skewed distribution.

THE ELUSIVE “AVERAGE”

MTTR is a common metric intended to provide a measure of “central tendency” for remediation efforts. Its virtues are simplicity and commonality, which aid comparisons of MTTR across different datasets or reports. But the data MTTR attempts to describe by a single point statistic is anything but simple. To understand why one need look no further than the heavily-skewed distribution in Figure 21 depicting the time flaws remain open after discovery.

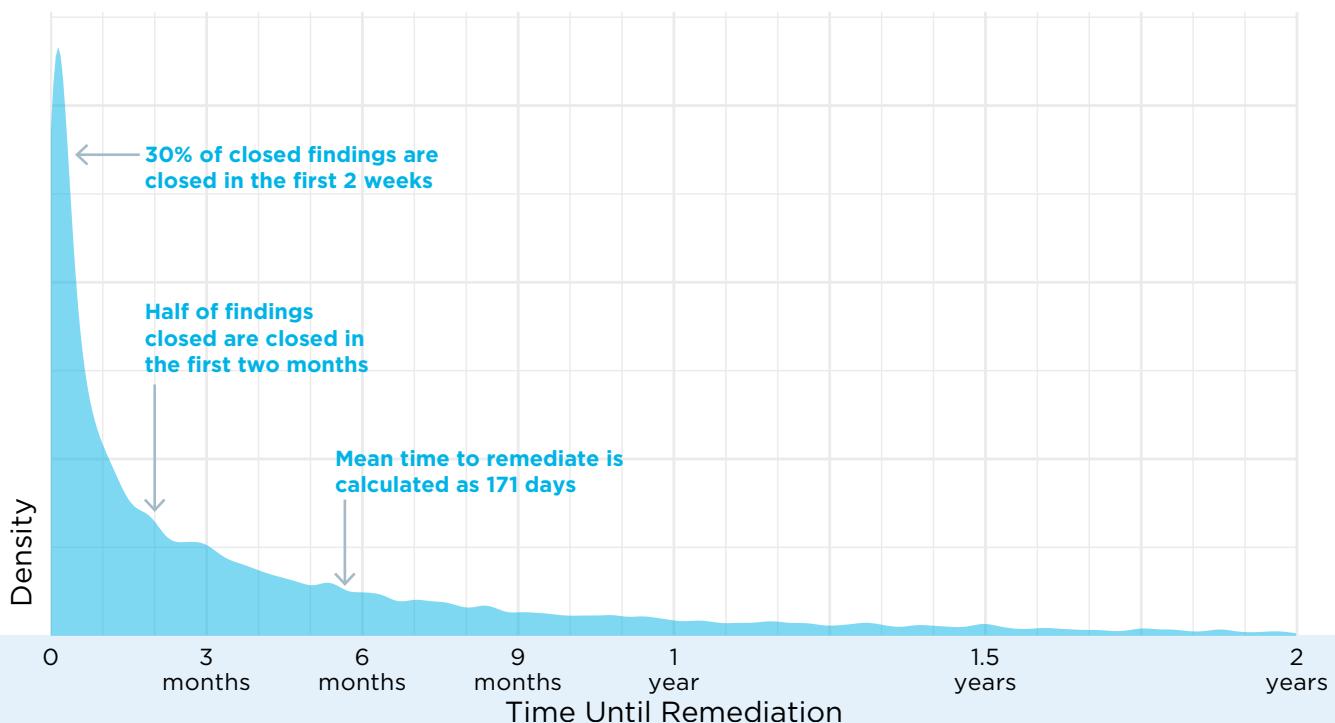


FIGURE 21 Distribution of time until remediation for closed flaws

Source: Veracode SOSS Vol. 10

Notice the huge spike on the left? When flaws are first discovered there's a rush of activity to fix them, but that effort is relatively short-lived. The excitement drops off quickly in the first month and continues to decline at a steady rate. Now step back and take in the full scope of remediation times in Figure 21. Where is the center or average? The large spike on the left makes it clear that closing flaws in under one month is the most common scenario by far. We can calculate the traditional average ("arithmetic mean") to get 171 days, but that measurement is heavily influenced by the long tail (the oldest flaw almost reached its 9th birthday before finally being addressed). By day 171, almost 3 out of every 4 findings are closed. That doesn't feel very "central", does it? We can also derive the median, which is just under two months (59 days). We'll hold onto that for now, because looking at closed findings is only half the story.

By focusing on just the time-to-remediation for closed flaws, we ignore all the unaddressed findings accumulating over time. These are the issues perpetually stuck in the "real soon now" status month after month. Per Figure 22, they show a much different distribution for remediation timeframe. Remembering that half of flaw closures in Figure 21 occurred in the first two months, now consider that this halfway point lies more than six months out. And roughly 10% of findings have been around for at least two years. That complicates any estimation of the average time-to-remediation, doesn't it?

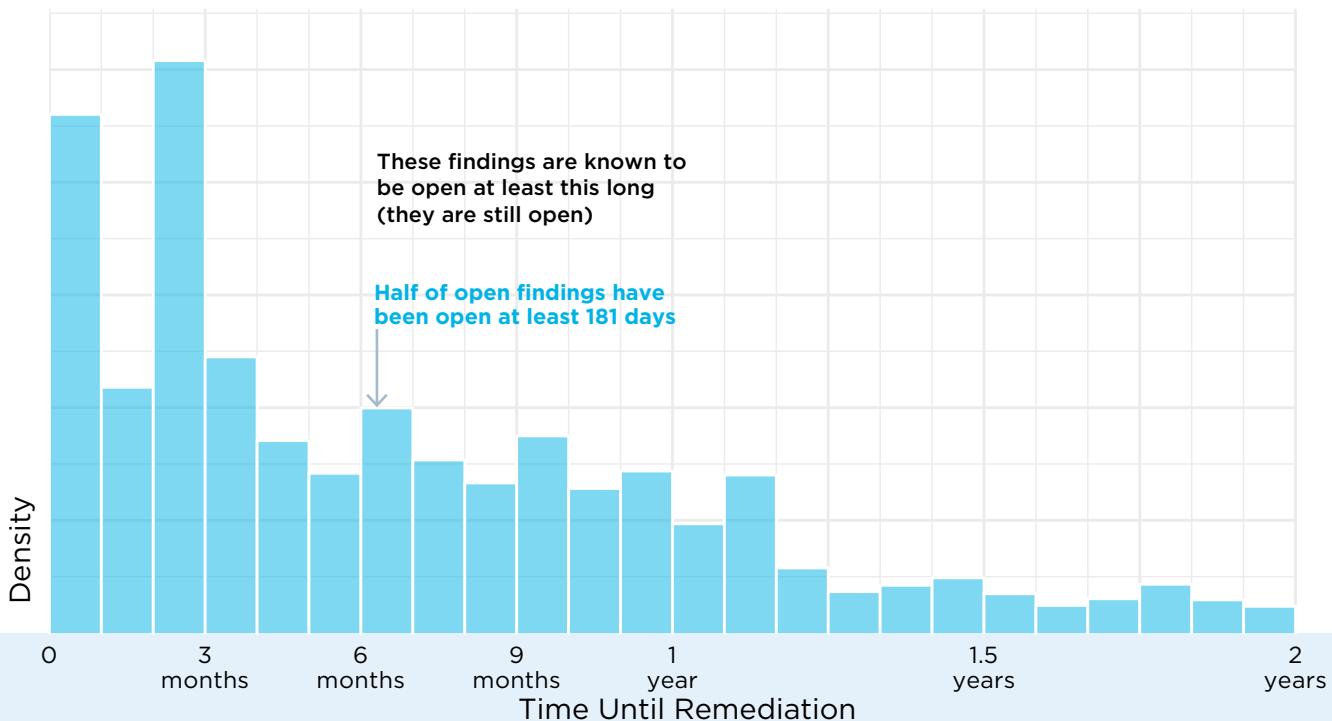


FIGURE 22 Distribution of time until remediation for all open and closed flaws

Source: Veracode SOSS Vol. 10

While there is no clear “Aha — there’s the average!” solution here, we can get one step closer by using both the timing of known closures and the knowledge that some findings have been open for at least some duration. Event analysis (also called survival analysis) takes both open and closed findings into account when calculating the probability that flaws discovered during our 12-month sample will “survive” the passing of time. SOSS veterans may recall we utilized survival analysis to study flaw persistence from numerous angles in Volume 9.

We leverage that technique in Figure 23, noting two new measures of centrality along the curve. The “closed median” of two months certainly doesn’t look like a measure of flaw half-life, and the “closed mean” also seems a bit optimistic from this perspective. The “event median” marks the point where survival analysis determines a flaw has a 50% chance of being remediated (eight months). That same technique places the expected lifespan of a finding, the “event mean,” way out beyond 20 months.

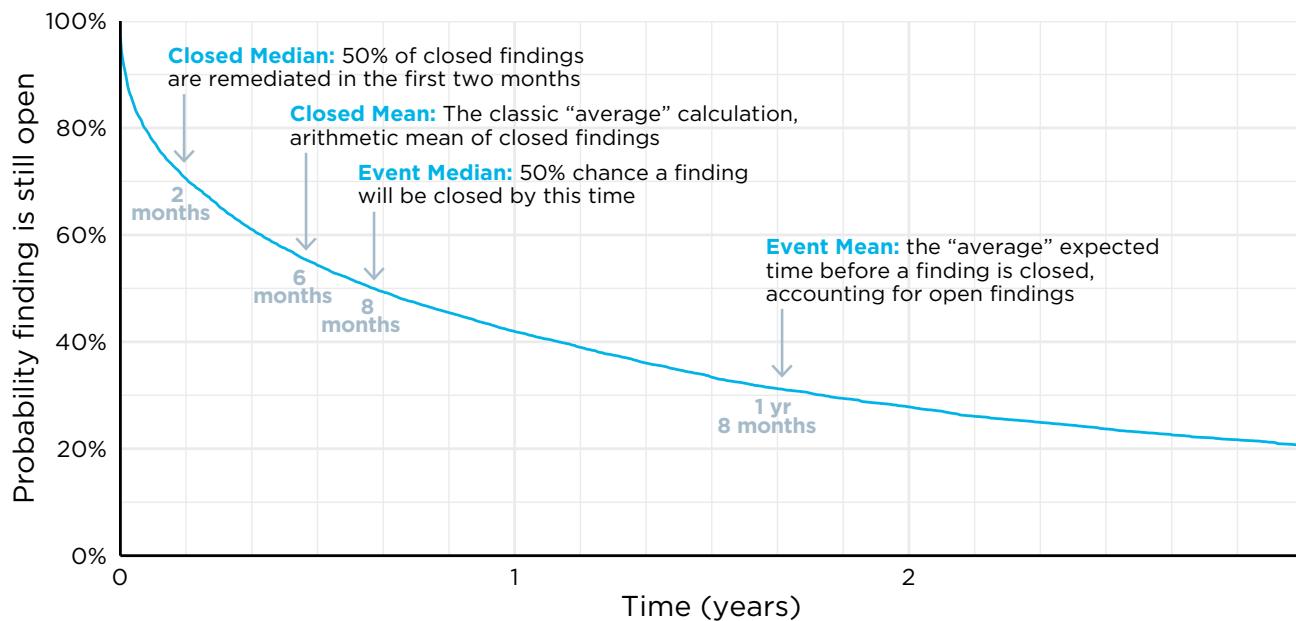
Again, it’s difficult to choose a winner from the four candidates for the coveted “average” time-to-remediation award marked in Figure 23. The “closed” stats have the strength of measuring what was actually fixed, but omit anything unfixed. The “event” stats capture the whole picture, but become increasingly inflated over time as unresolved flaws build up. Both are true, but neither story tells the whole truth.

Unless otherwise noted, we use the “Closed Median” or “MedianTTR” for time-to-remediation statistics in this section. We do so because it is the most appropriate and direct option given what we want to measure — a highly skewed distribution of time required to remediate flaws (that were actually closed).

FIGURE 23

Flaw persistence curve with four possible central statistics for time-to-remediation

Source: Veracode SOSS Vol. 10



Which flaws are fixed the fastest?

Now that we have a better working definition of what's "average" (or median, rather) for remediation time frames, we're ready to slice and dice this metric across the various segments. We'll start from the beginning — the supplier or source of the application itself — in Figure 24.

We recommend anchoring your review of Figure 24 on the 72 days indicated as the median remediation timeframe for internally developed applications. It's not shown but worth noting that insourced software posted the highest fix rates. Software supplied by contractors or open sourced take about a month longer than that. Interestingly, an outsourced team gets the job done a couple weeks faster.

Since the source of an application appears to affect fix speed, it seems logical that its criticality to the business might as well. Figure 25 doesn't cause us to reject that hypothesis, but doesn't overwhelmingly support it either. Barring one exception for "High," the MedianTTR does indeed shorten somewhat as criticality rises. The lower and upper quantiles, however, display no such pattern, and the least critical systems hit the 75% remediated mark months before anything else. Clearly, other factors must be in the mix.

Figures 24-27 mark the MedianTTR with blue dots. Imagine those are the end of a standard bar chart if you're just looking for a simple comparison among items. We add the 25% and 75% closed marks to indicate the large amount of variation around those medians.

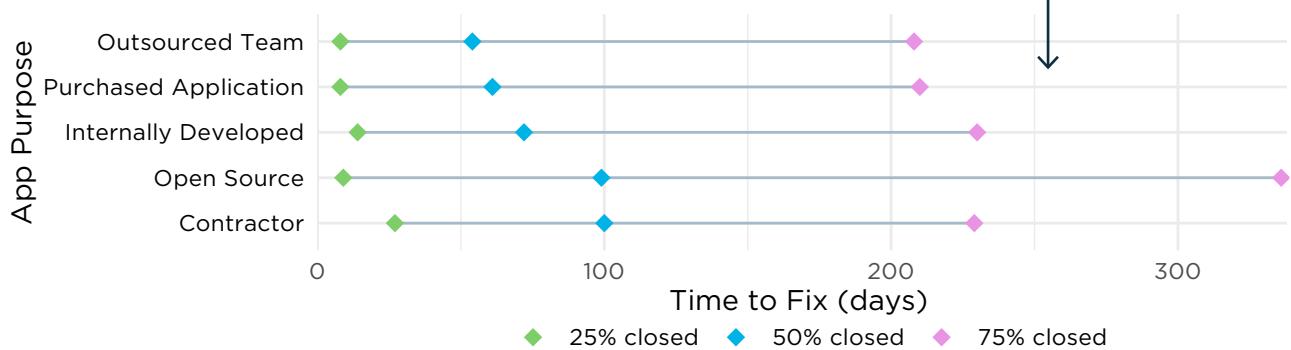


FIGURE 24 Comparison of time-to-remediation statistics across application sources

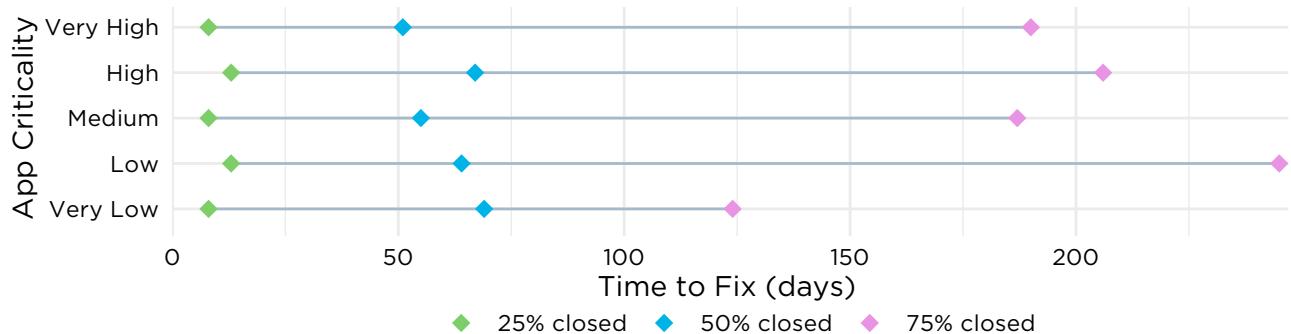


FIGURE 25 Comparison of time-to-remediation statistics across application criticality ratings

Source: Veracode SOSS Vol. 10

Perhaps the flaw severity score is one of those factors. After all, the stated purpose of assigning these scores is to help organizations prioritize remediation efforts on security findings that pose the highest risk to the application. And according to the results observed in Figure 18 for fix rates, those recommendations do hold weight. We see that effect here as well.

Flaws considered to be of the very highest severity get addressed most expediently, according to Figure 26. That holds true right out of the gate (25%), midway through the race (50%), and in the home stretch (75%). Other severity levels aren't quite so unwavering in that regard, but there's a fairly consistent trend of quicker fixes of more severe flaws through the 25% and 50% closed waypoints. That focus appears to fall apart late in the game, but we have a theory for what's going on there that we'll get into later.

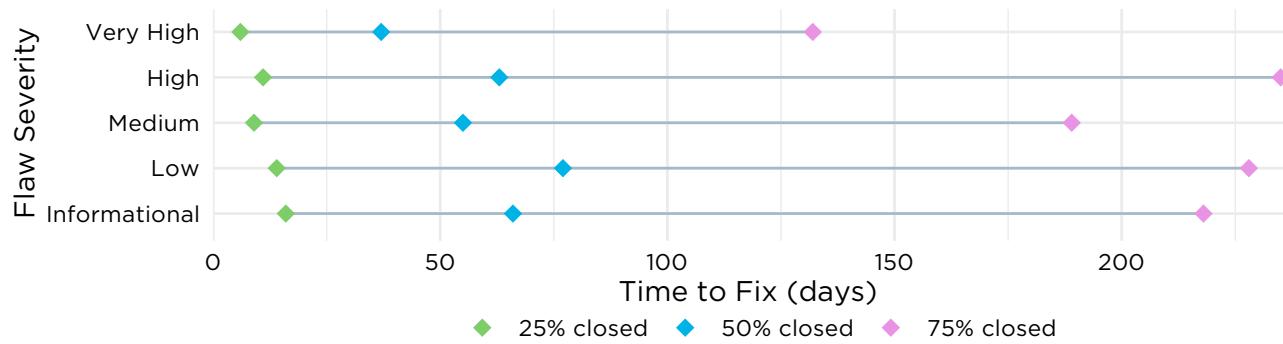


FIGURE 26 Comparison of time-to-remediation statistics across flaw severity scores
Source: Veracode SOSS Vol. 10

Moving on to Figure 27, we see fix time statistics for specific types of flaws. Before commenting on the flaws themselves, though, let's take in the big picture. The first big takeaway is that a high degree of variation exists among flaw categories. The MedianTTR for flaws at the bottom of the list is 5X longer than those at the top. Differences are even more dramatic at the 75% closure mark, with hundreds of days separating the fastest and slowest categories. Some of this variation stems from perceptions about which flaws are the most important to fix quickly or which represent the most risk. But these results also undoubtedly reveal developers taking action on what's easiest for them to fix, which may not necessarily align with what's most important.

Much of what we see here matches up with our intuition as security professionals. Flaws such as Authentication/Authorization issues, Code Injection, and Credential Management are the types of flaws likely to draw the ire of internal security teams. Deployment Configuration, Server Configuration, and the like are infrastructure-specific issues that can be changed more easily than a complex application code base, contributing to the relatively good fix times for these issues.

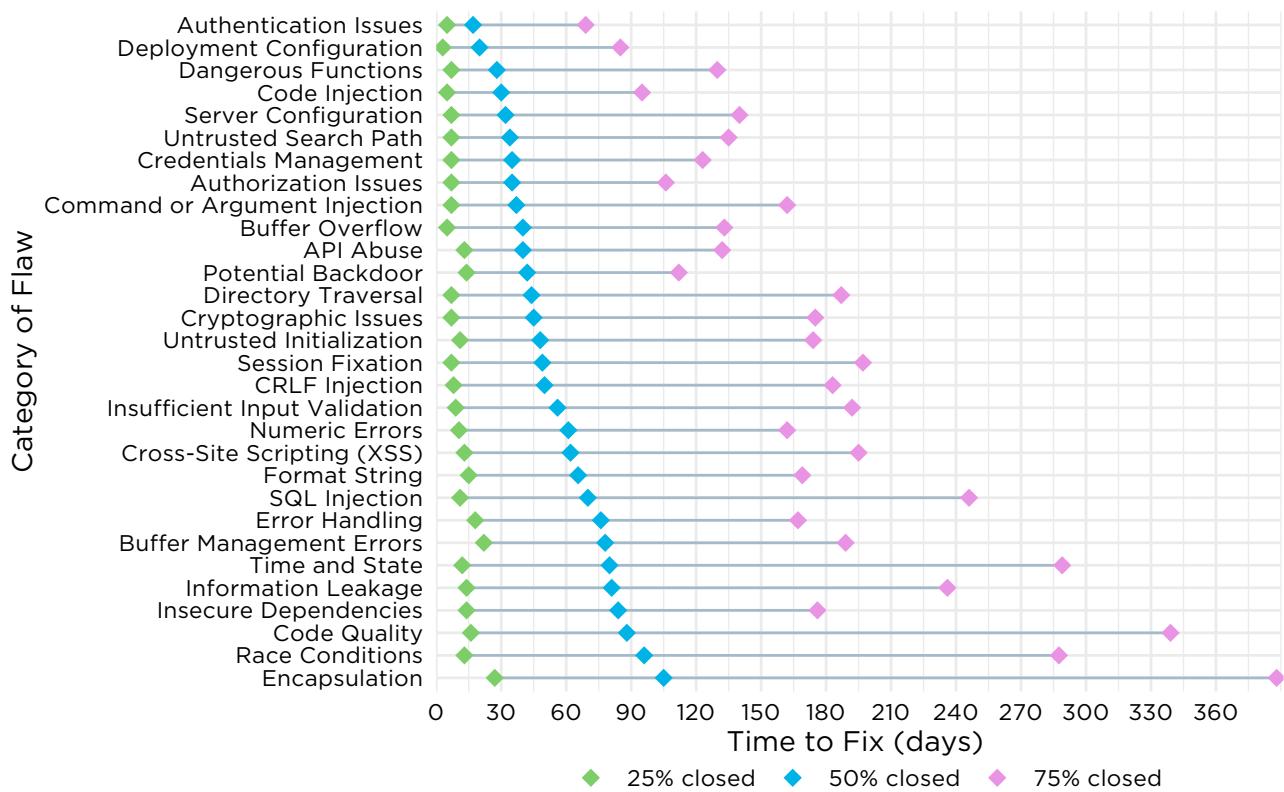


FIGURE 27 Comparison of time-to-remediation statistics across flaw categories

Source: Veracode SOSS Vol. 10

Figure 28 combines two important aspects of fixing flaws — comprehensiveness (fix rate) and speed (MedianTTR). We believe it to be a useful comparison because it categorizes the urgency with which flaws are addressed. Those fixed widely and quickly are “Prioritized” by developers, whereas those largely left open for long periods of time are “Neglected.”

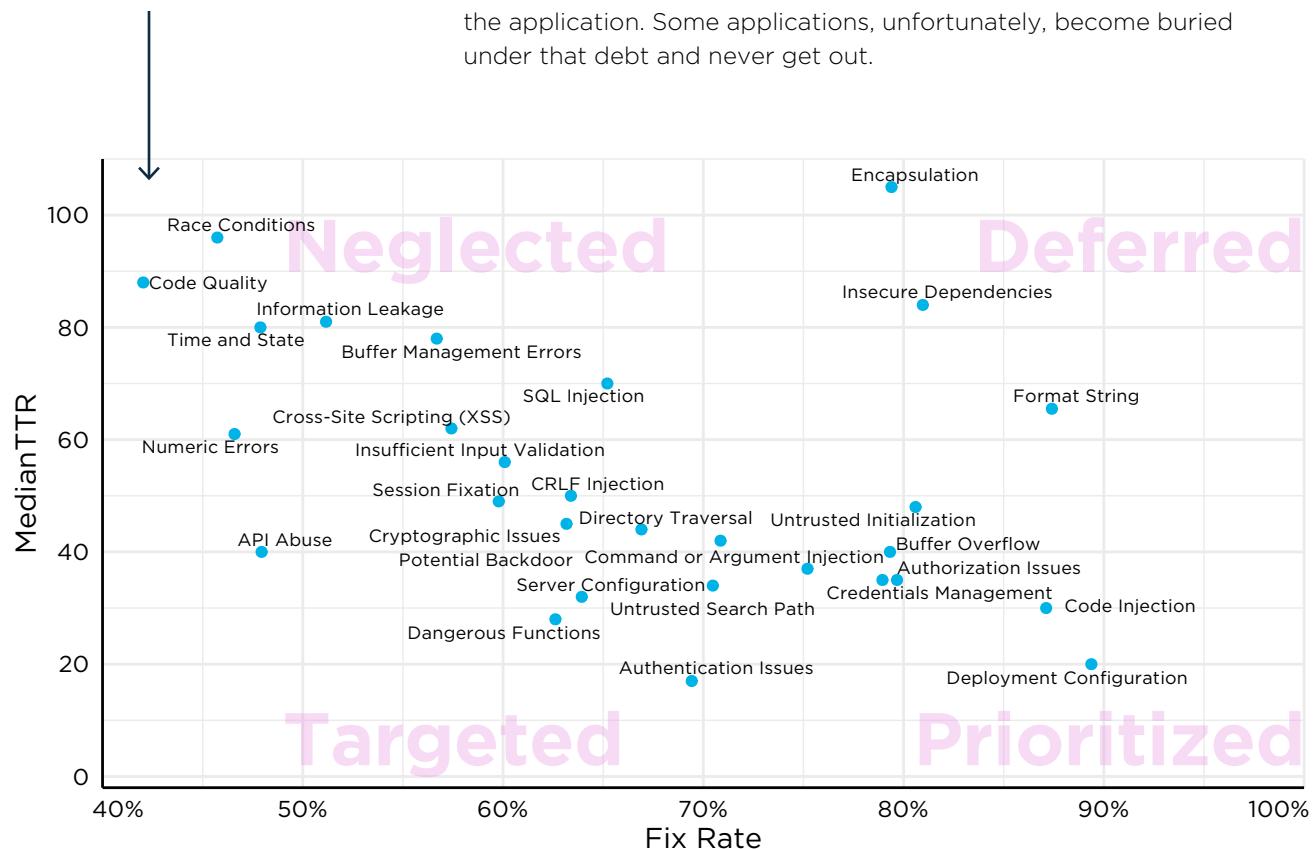


Figure 28 offers an interesting view combining fix rate and MedianTTR for flaw categories. We've already discussed each of those measures individually and so will restrict our commentary here to the labels superimposed over the four quadrants of the grid.

Flaws listed in the lower right quadrant are fixed both comprehensively and quickly. Thus, we can reasonably conclude they receive priority attention from developers due to perceived importance, ease of fix, or some other reason. Those in the upper right usually get addressed, but not until higher-priority issues are dealt with first.

The lower left quadrant is sparse, but flaws near that region tend to be remediated fairly quickly on a small scope. A plausible scenario for this might be a flaw that affects a critical component within an application. The organization may choose to remediate the flaw in that component but not (yet) across the entire application. This is what we mean by “targeted” remediation.

Flaws in and around the upper left of the grid receive neither comprehensive or quick attention. Perhaps there's a legitimate reason for neglecting them in some cases; we've mentioned a few of those earlier in this report. But ignored long enough, these findings become security debt, gradually accumulating over the lifespan of the application. Some applications, unfortunately, become buried under that debt and never get out.

FIGURE 28 Fix rate and MedianTTR for flaw categories

Source: Veracode SOSS Vol. 10



Breaking Down Security Debt

The concept of security debt has come up several times so far in this report. We've seen that unaddressed flaws don't simply disappear from applications, but rather accumulate over time. To combat this, we need to better understand the mechanics and makeup of software security debt. That's the purpose of this final section.

Do priorities contribute to security debt?

Figure 29 tracks the probability of a flaw being fixed over time. It shows that younger flaws (those discovered recently) are more likely to be fixed than older flaws, and hints at a “recency bias” at work in flaw remediation practices. These unaddressed older flaws accumulate over time and become security debt in applications.

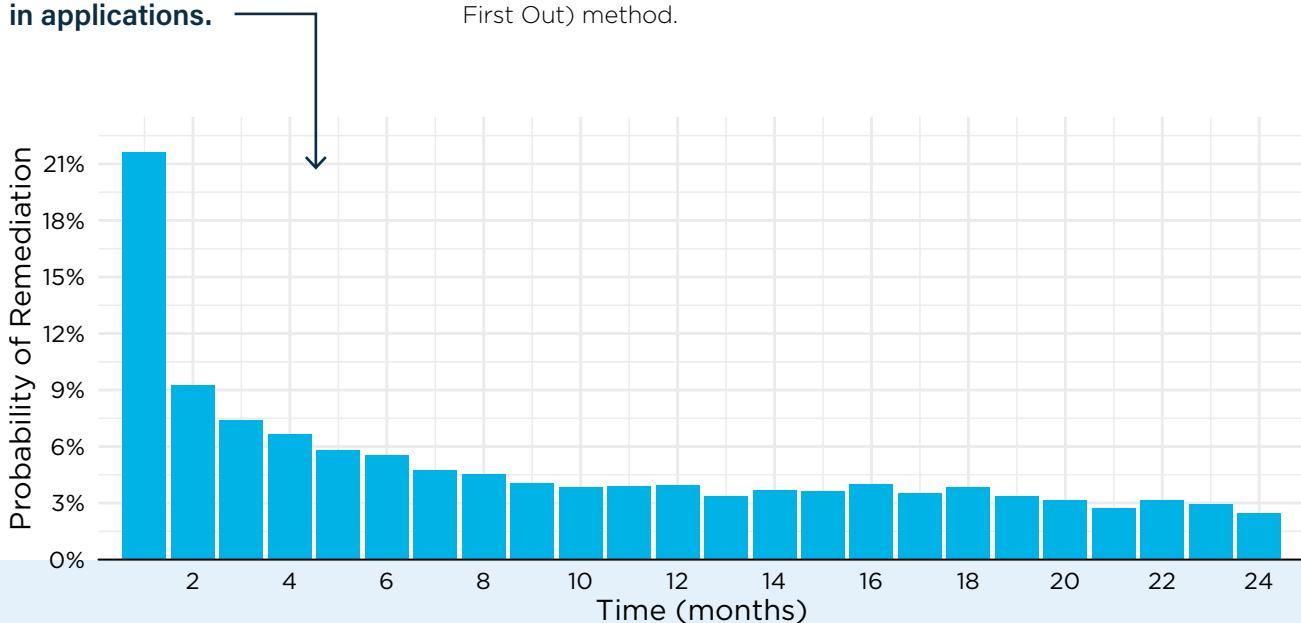


FIGURE 29

Monthly probability of remediation based on flaw age

Source: Veracode SOSS Vol. 10

What do you do with that information? For starters, it never hurts to know how biases affect behaviors. In terms of changing those behaviors, neither LIFO or FIFO methods seem optimal for processing findings. We'd like to see more of a PIFO (Priority In, First Out) approach where any debt that accumulates consists only of inconsequential flaws. But that's not the way things appear to work in practice. Reality suggests there's a capacity element involved in the debt equation in addition to prioritization.

Is there a security debt-to-income ratio?

We use “income” here to refer to the working capacity development teams have available to allocate to flaw remediation. If their capacity to fix flaws consistently exceeds the rate at which flaws are introduced to the codebase, security debt in applications should go down over time. If flaw creation exceeds capacity, we should see the opposite trend.

Earlier sections of this report hint that not all development teams possess sufficient capacity to reduce the number of flaws in their applications. The overall fix rate, for instance, stands at 56%, meaning teams close a little over half of the findings they discover. We also learned that 30% of applications showed a buildup of flaws between their first and last scans for the sample period. Figure 30 goes deeper in that line of analysis, presenting a comparison of open and closed flaws across tens of thousands of applications during our sample period.

The blue dots in Figure 30 represent applications with a positive fix capacity (they fixed more flaws than they found) and the fuchsia dots mark those with a negative capacity (found more than they fixed). Any application forming the dark line in the center maintained a steady balance. Not shown are the 25% of applications that remained flaw-free for the whole year.

Figure 30 presents a comparison of open and closed flaws across tens of thousands of applications during our sample period. Blue dots represent development teams gaining ground against flaws and the fuchsia dots mark those falling behind.

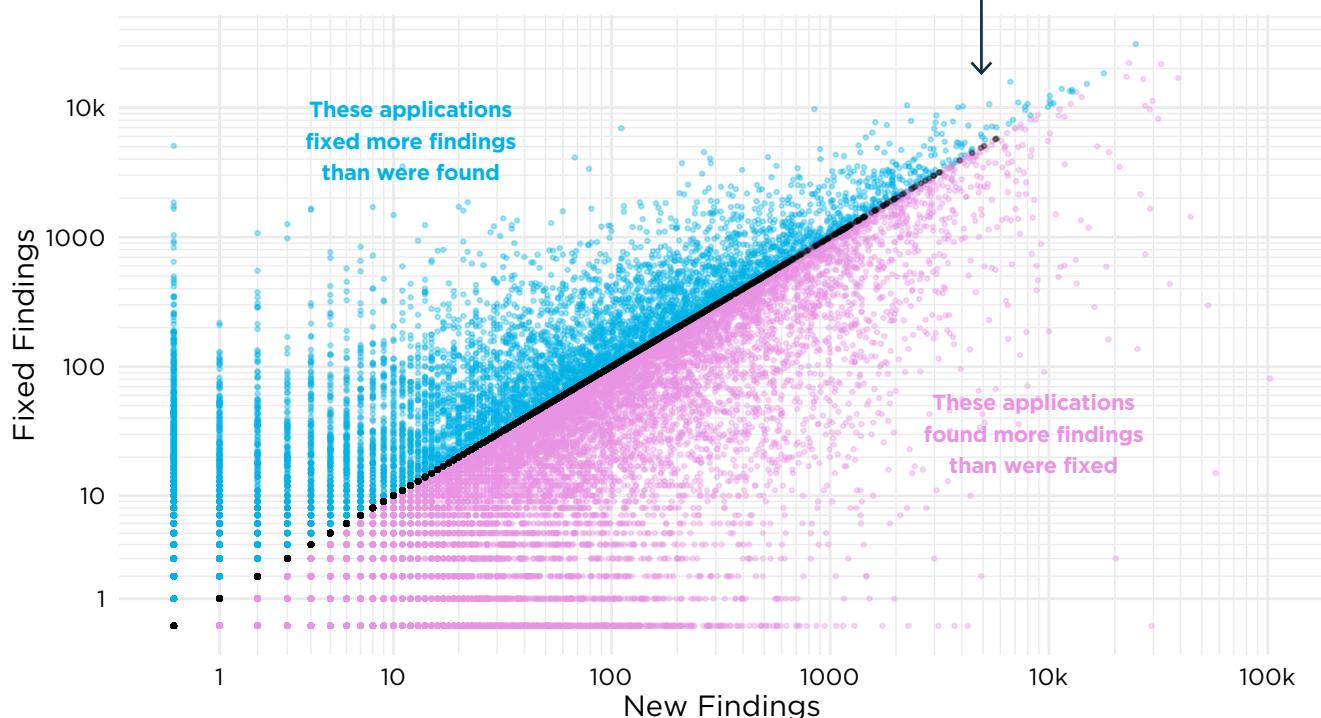


FIGURE 30 Ratio of discovered vs. remediated findings among applications

Source: Veracode SOSS Vol. 10

Aside from being pretty, the plot highlights the extreme amount of variation in the ratio of discovered versus remediated findings. Applications cluster toward the breakeven line, but there are some extremes on either side. Figure 31 makes that contrast easier to see and compare. About half of development teams are falling behind flaws in their applications and a little over a quarter gaining positive ground.

The two sides in Figure 31 don't sum to 100% because we've removed approximately one-quarter of applications that maintained a zero balance for security debt over the year. It's the two sides of the "mountain" that we're interested in here and the extreme amount of variation we see in terms of fix capacity. The majority of applications fall within a +2X capacity, but some fix (or discover) 10X, 20X, or even 100X the number of flaws they discover (or fix)!

The two sides in Figure 31 could be seen as a comparison between the "haves" (top) and the "have-nots" (bottom). But it could also be argued that an application's place in Figure 31 is less about having vs. not having and more about setting priorities. Namely, does the business invest more heavily in reducing security debt or reducing time-to-market? Either way you choose to interpret it, one thing is clear; there are many application teams on both sides of that line.

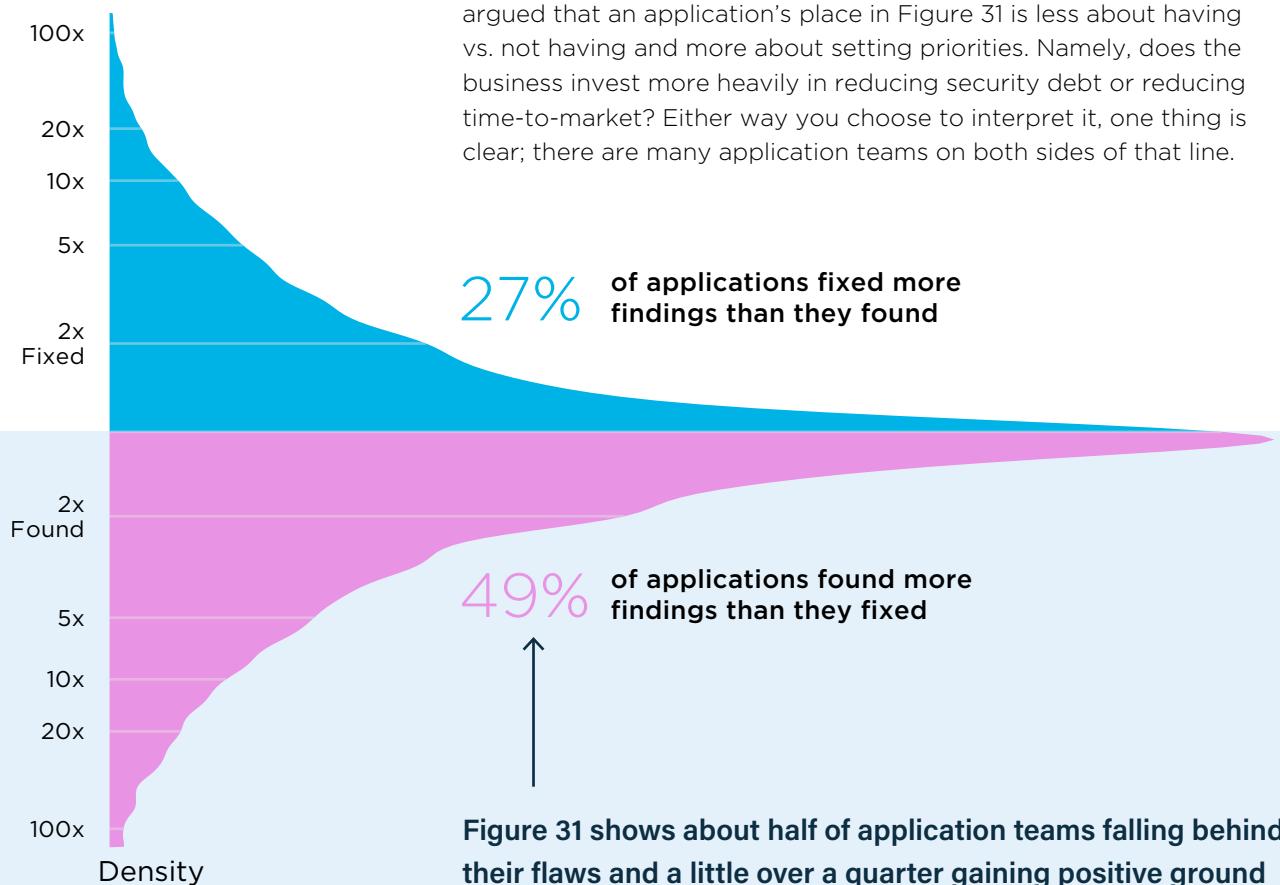


FIGURE 31

Proportion of applications reducing security debt (top) vs. adding debt (bottom)

Source: Veracode SOSS Vol. 10

Is fix capacity a constant?

We've already seen that the proportion of applications that increase vs. decrease or maintain security debt is relatively even. Now we want to know if that fate is locked in or if it changes under certain conditions. Let's start by generating a data-driven picture of what abstract concepts like fix capacity and security debt look like. Figure 32 does just that.

Figure 32 is meant to model how a typical application pays down and accumulates security debt over time. The dark blue bars at the top correspond to weekly flaw closures. It's rather obvious that, on average, only a small proportion of known issues are successfully closed at any given time. The pink area tallies the average number of unresolved findings each week, taking into account the preexisting balance, new flaws found, and old flaws closed.

The fact that Figure 32 bears an uncanny resemblance to an iceberg did not escape us, and we're pretty sure it made a similar impression on you. Though development teams focus heavily on keeping up with new flaws they discover in applications, that mountain of aging security debt looming under the surface cannot be ignored.

Figure 32 models the mechanics of security debt in a typical application. The dark blue bars on top correspond to weekly flaw closures. The pink area tallies the average number of unresolved flaws carried over each week. Though development teams focus on keeping up with new flaws, the mountain of security debt looming under the surface cannot be ignored.

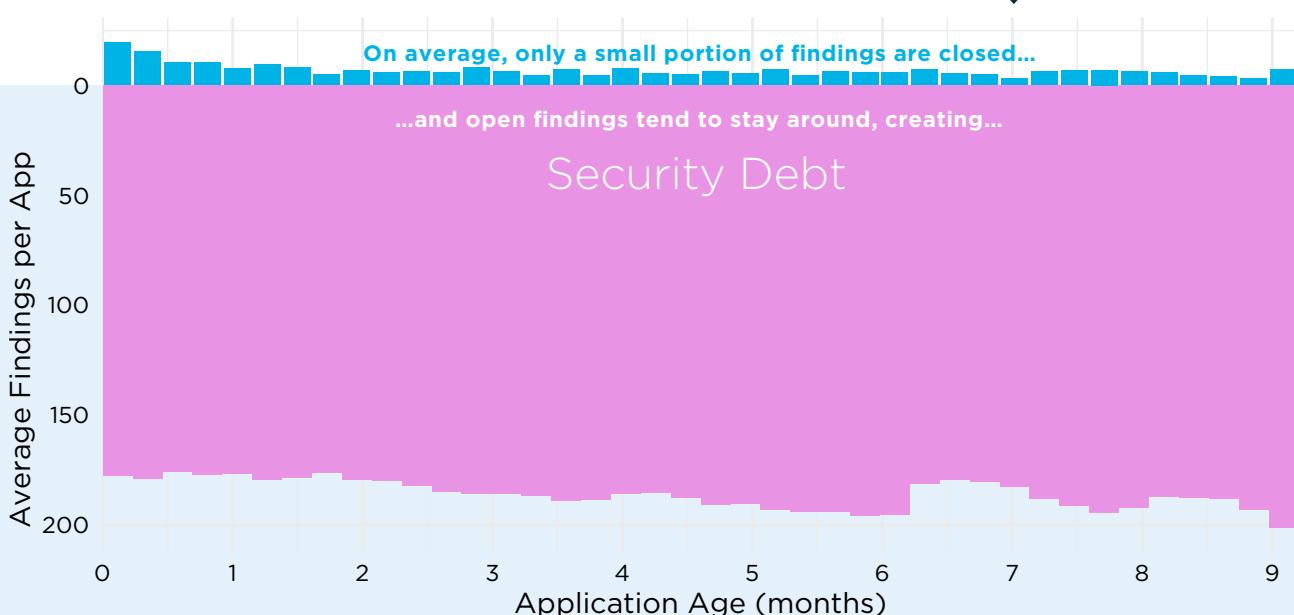


FIGURE 32 Depiction of flaw closure (fix capacity) and accumulation (security debt) over time

Source: Veracode SOSS Vol. 10

This brings up an interesting dilemma about how to accomplish two difficult feats: staying on top of new flaws introduced during the development process while chipping away at security debt littering the codebase. The good news is that evidence shows it can be done. The ability to accomplish this feat seems to be partially about who you are, partially about what language you use, and partially about how you integrate security into the development process.

Let's start with who you are. Figure 33 compares the typical remediation and debt cycle for different industries. We recommend not getting too caught up in the minor details and movements here. Note instead the average size of security debt for each sector and whether that's growing or shrinking over time. The main takeaway here is that some industries seem more or less prone to security debt than others.

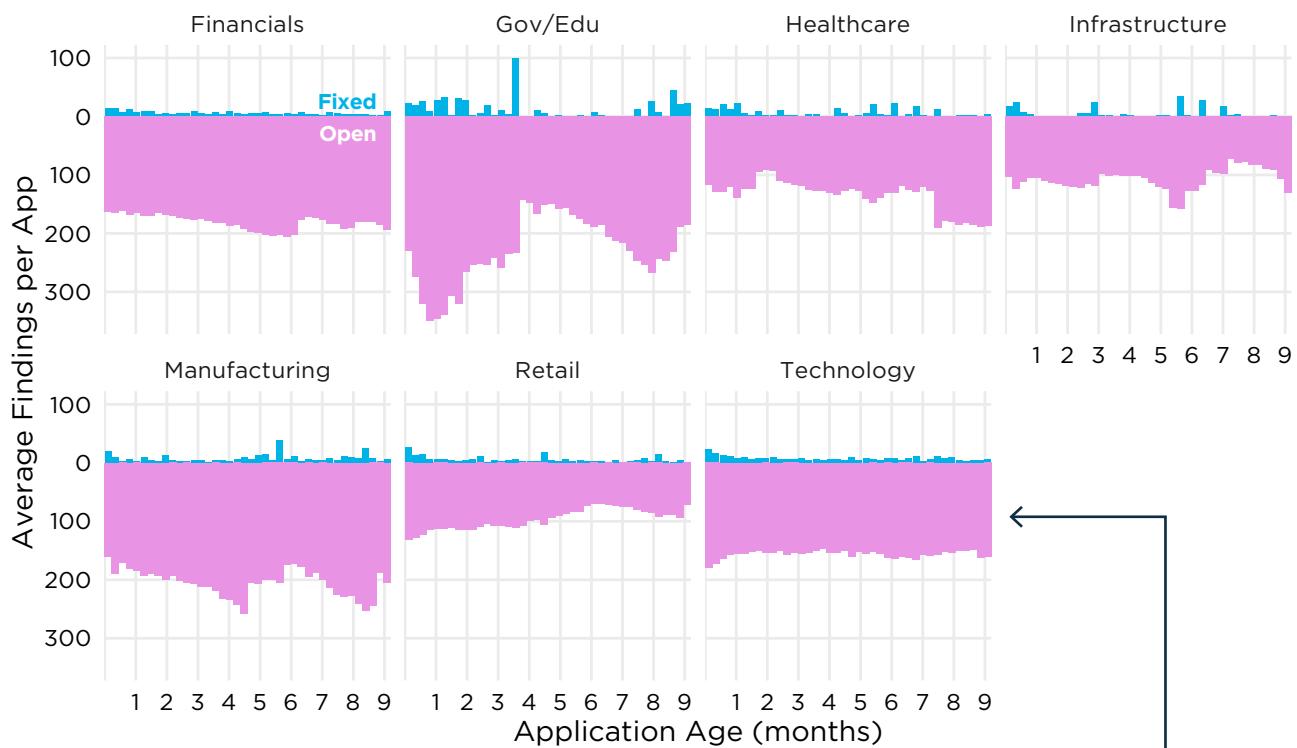


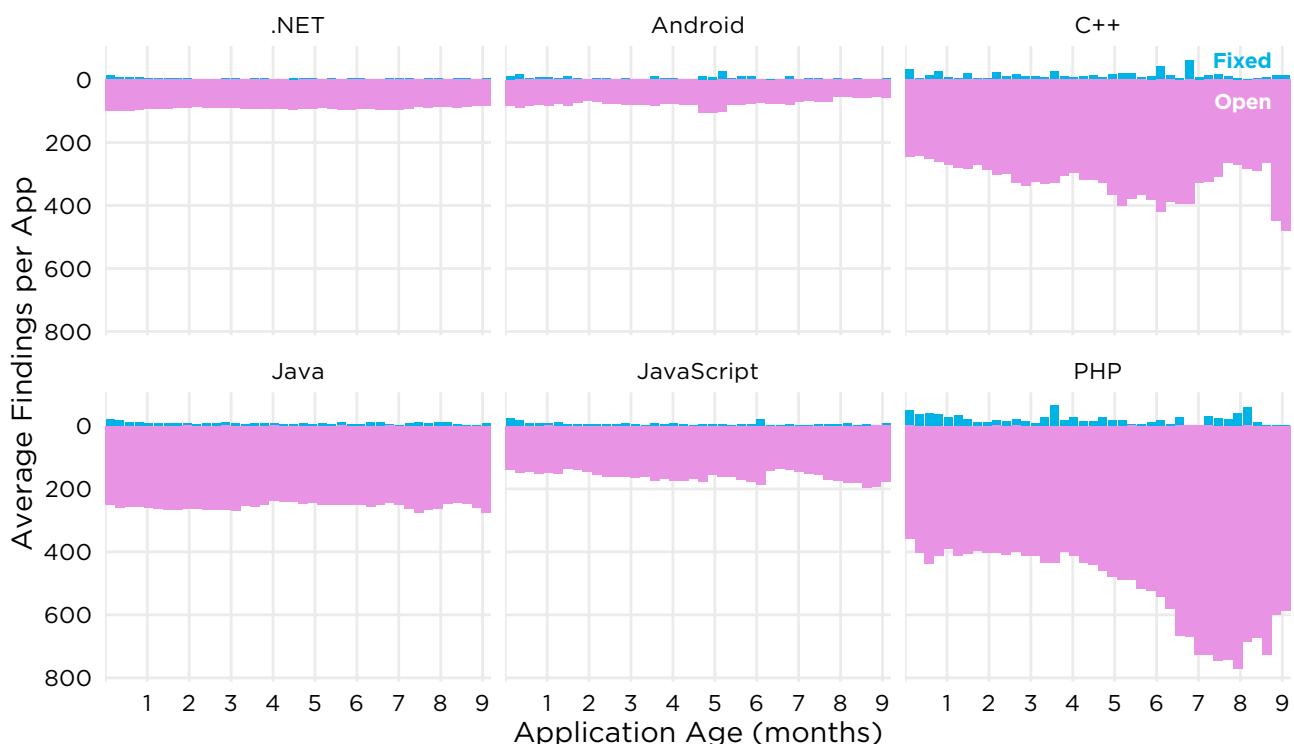
FIGURE 33

Comparison of fix capacity and security debt by industry

Source: Veracode SOSS Vol. 10

We recommend not getting too caught up in the minor details of Figures 33-37. Note instead the size of security debt (icebergs) for each category and the trend over time. The main takeaway is that some categories seem more/less prone to security debt than others.

Moving on to what language you use, Figure 34 presents a view of security debt split across the top programming languages identified among applications tested. The differences here are surprisingly dramatic (notice the scale on the vertical axis). Average security debt for PHP apps dwarfs everything else and C++ carries a debt that's anywhere from 3X to 5X larger than .NET at various points over the sample period.



The point here is not "Stay out of debt by ditching C++ for .NET," though that does make for a rather nice jingle. We realize that most teams can't up and change what language they're using on a whim and do not recommend such a drastic course of action. What we do recommend taking away from Figure 34 is an awareness that some applications may be more or less prone to the buildup of security debt irrespective of anything else you do. With that awareness, you can consider viable actions to take in order to counter and control language-based proclivities to security debt.

FIGURE 34

Comparison of fix capacity and security debt by application language

Source: Veracode SOSS Vol. 10

The top 1% of applications with the highest scan frequency carry about 5X less debt than the bottom one-third.

Speaking of actions you can take, Figures 35 and 36 examine the effect of scanning frequency and cadence on security debt. As we saw with fix rates and remediation speeds, results indicate that the way in which development teams approach these activities can have a positive impact. The top 1% of applications with the highest scan frequency carry about 5X less debt than the bottom one-third.

In Figure 35, we see the total volume of debt reducing steadily with more frequent security scanning. That said, the most frequently tested applications show a buildup of security debt over the period. We suspect this is partly due to the relatively low number of applications in that category. But it may also be that these applications were scanned so often because they were in an active development phase. The main point is that the propensity to accumulate debt is much less when applications undergo frequent testing.

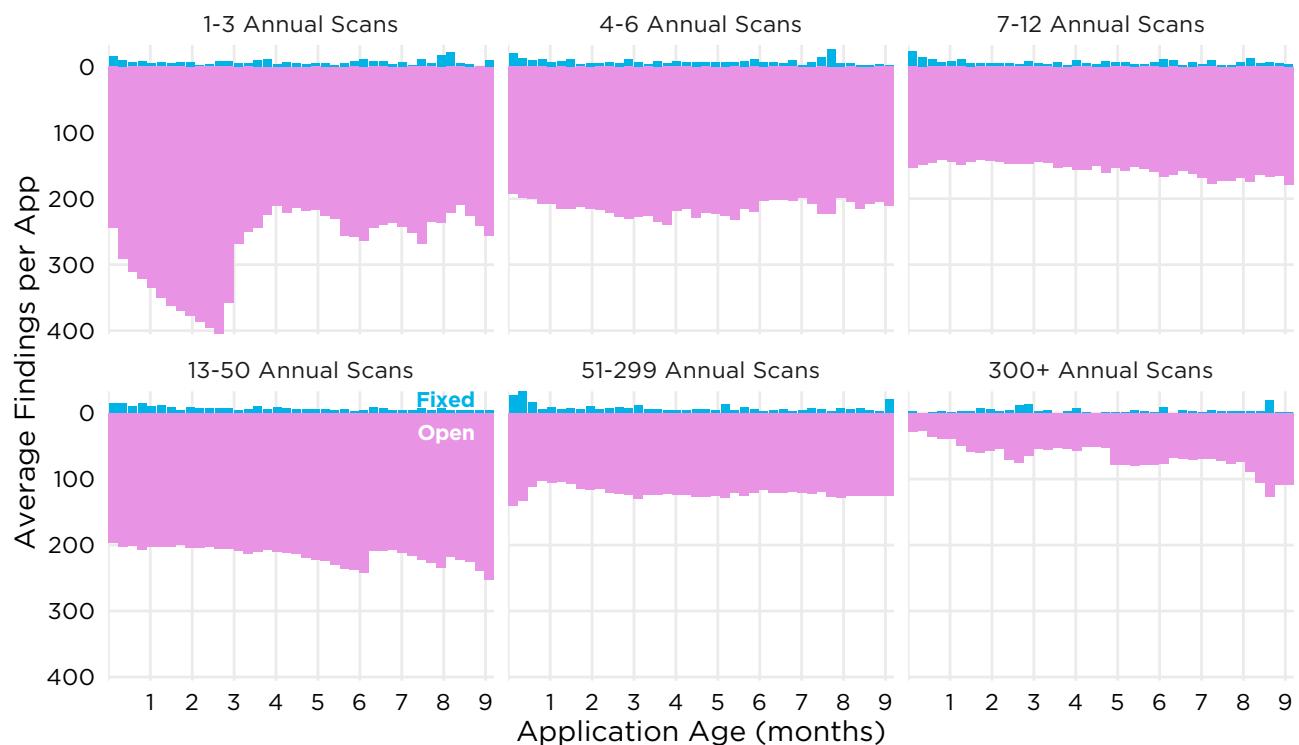


FIGURE 35 Comparison of fix capacity and security debt by scan frequency
Source: Veracode SOSS Vol. 10

The relationship of scanning cadence to security debt depicted in Figure 36 isn't as obvious. For most of the time period studied, applications scanned at the various intervals carried similar amounts of debt (though a 'Steady' cadence averaged about 30% less). Where they ended, however, is a different matter. Steadily tested applications began chopping away at their end-of-year security debt, while those subject to bursty scanning began rapidly piling it on. This may again be attributable to small-ish numbers. But it's also logical that applications in the bursty category entered a rapid development phase before an offsetting burst of scans was conducted.

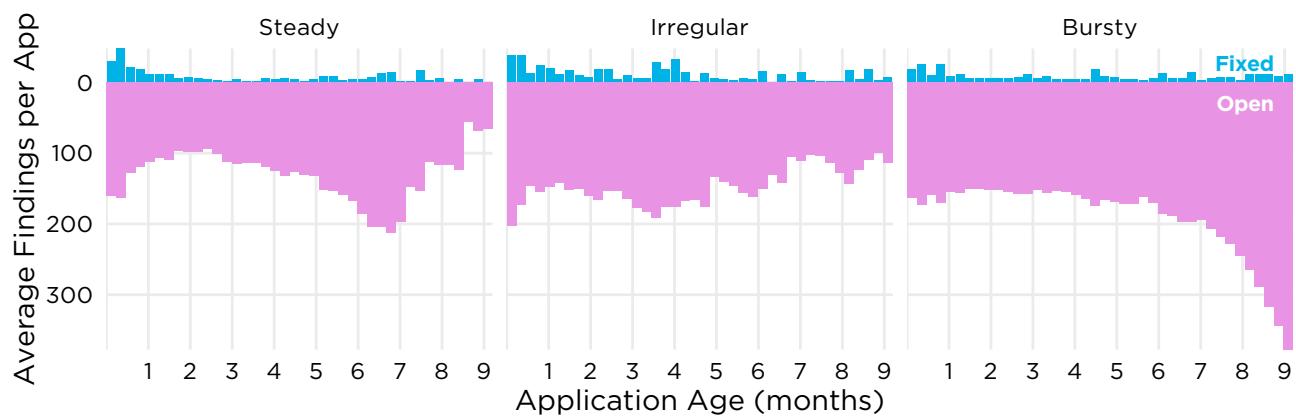


FIGURE 36 Comparison of fix capacity and security debt by scan cadence

Source: Veracode SOSS Vol. 10

The largest amount of debt across applications comes from Cross-Site Scripting (XSS).

What is security debt comprised of?

If these icebergs of debt are lurking beneath the surface of our applications, what sort of flaws make up this debt? We start by examining the long-unaddressed OWASP Top 10 flaws displayed below in Figure 37. Interestingly, the composition of flaws does not match the overall prevalence we saw back in Figure 19, suggesting that certain types of flaws are more likely to become security debt than others. The largest amount of debt across applications comes from Cross-site Scripting (XSS), with Injection, Authentication, and Misconfiguration flaws making up sizable portions as well. We consider this noteworthy, as Injection is the second most prevalent flaw category in reported exploits (recall Figure 20).

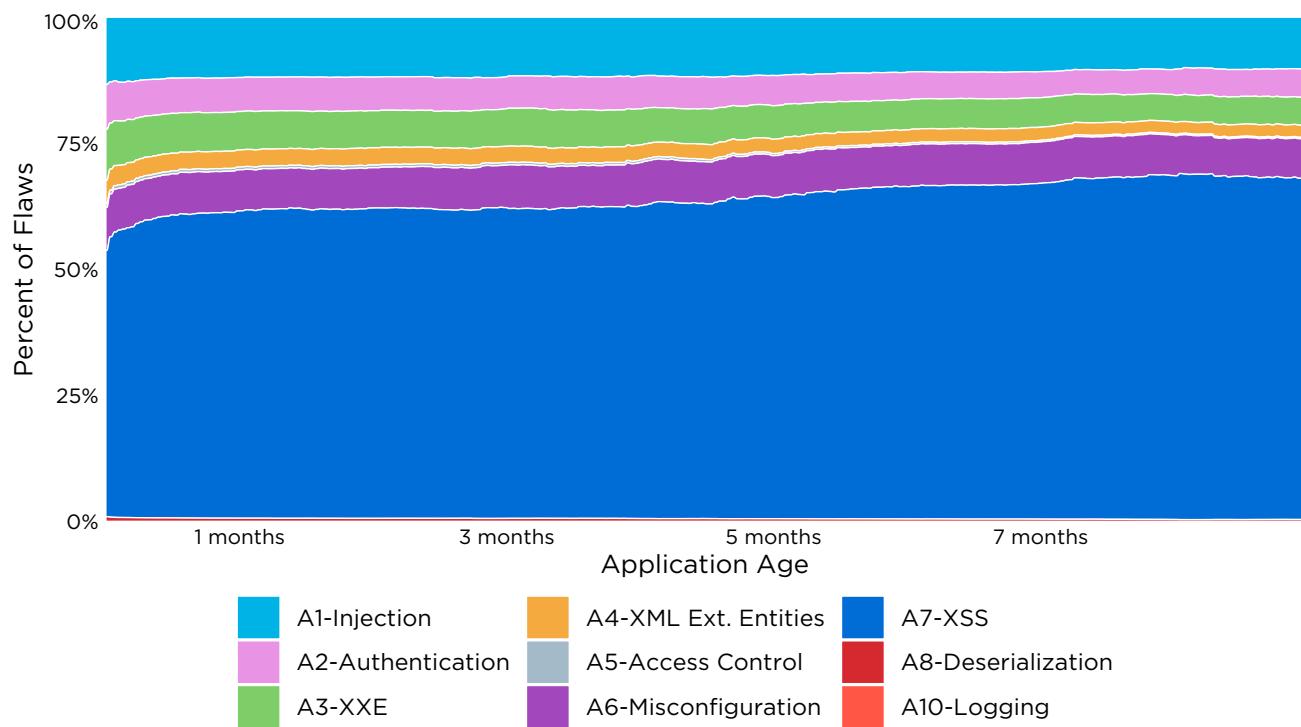


FIGURE 37 Percentage breakdown of flaw debt over the lifetime of an application

Source: Veracode SOSS Vol. 10

Perhaps more interesting is the breakdown of debt by language presented in Figure 38. We exclude the predominant XSS category to get a better look at the less common flaws across languages. For JavaScript we see lots of security debt in the form of Injection and Authentication flaws. Misconfiguration errors are largely concentrated in Java, with large portions in .Net and PHP as well. C++ stands out as the only language with a significant portion of Access Control flaws and with debt that spans a variety of flaw types.

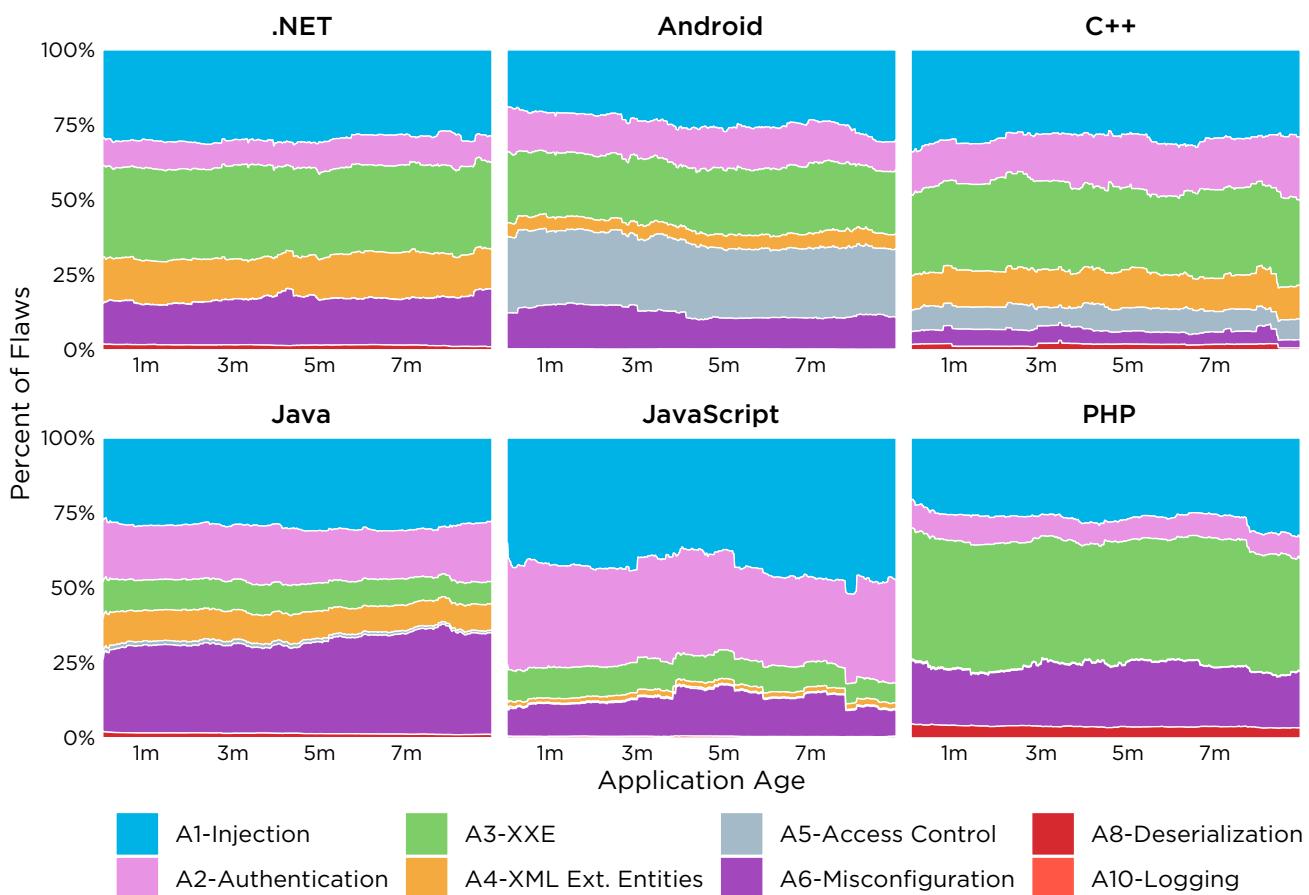


FIGURE 38 Flaw debt types across application age by language

Source: Veracode SOSS Vol. 10

A steady scanning cadence is the only cadence associated with meaningful change in the proportion of flaw types.

Before we leave the land of OWASP breakdowns, we want to see if scanning cadence affects the type of flaw debt found in an application and whether that changes over the application's lifetime. The results are below in Figure 39. A steady scanning cadence is the only cadence associated with meaningful change in the proportion of flaw types. Specifically we see a steady reduction in Authentication flaw debt as the drumbeat of scans progresses. This is positive as Authentication flaws are most likely to be used in incidents (Figure 20).

A bursty or irregular cadence does not appear to significantly change the nature of security debt as the application ages. We're encouraged to see the proportionality of flaws change with steadier scan patterns. We suspect that this reflects teams using their scan results in a more mature process whereby they choose the types of issues to tackle first and the types of debt they're willing to tolerate. That puts them closer to being on top of their security debt rather than drowning under it. And that's a much better place to be.

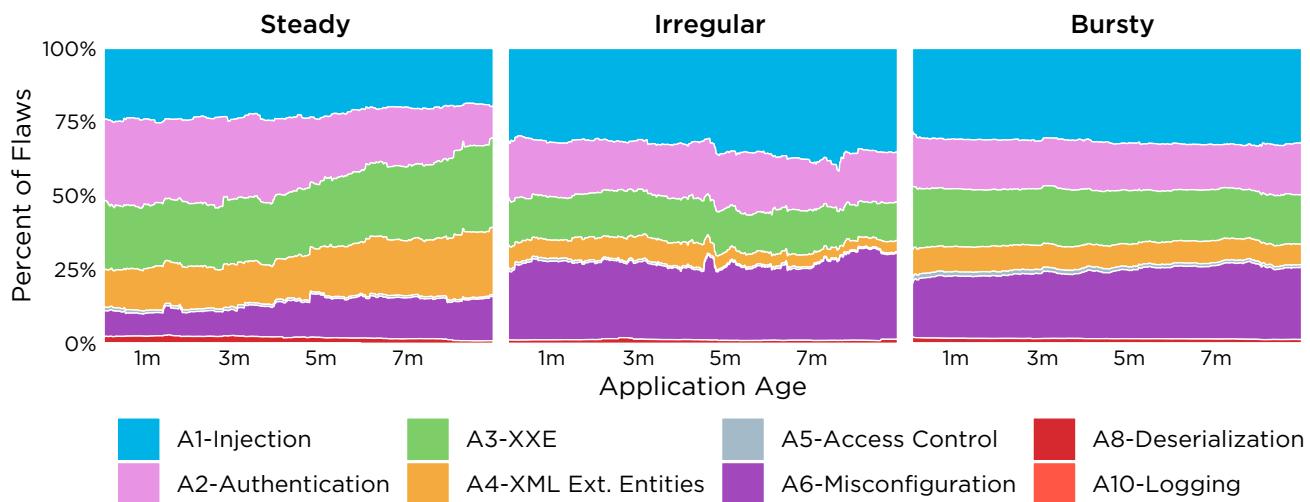


FIGURE 39 Flaw debt type by scanning cadence

Source: Veracode SOSS Vol. 10



Regional Breakouts

While our dataset associates applications with the geographic location of the organizations to which they belong (as opposed to, for instance, the location of development teams), we can still glean some interesting regional trends.

Does software security change by region?

Figure 40 compares the three high-level regions according to several key measures from our software security testing over the last year. Proceeding from left to right, the columns shed light on the mechanics of security debt, starting with the proportion of applications with higher-severity (level 4 or 5) flaws, the percentage of those flaws that are fixed, the median speed at which those flaws are fixed, and the average number of unfixed flaws (debt) per application.

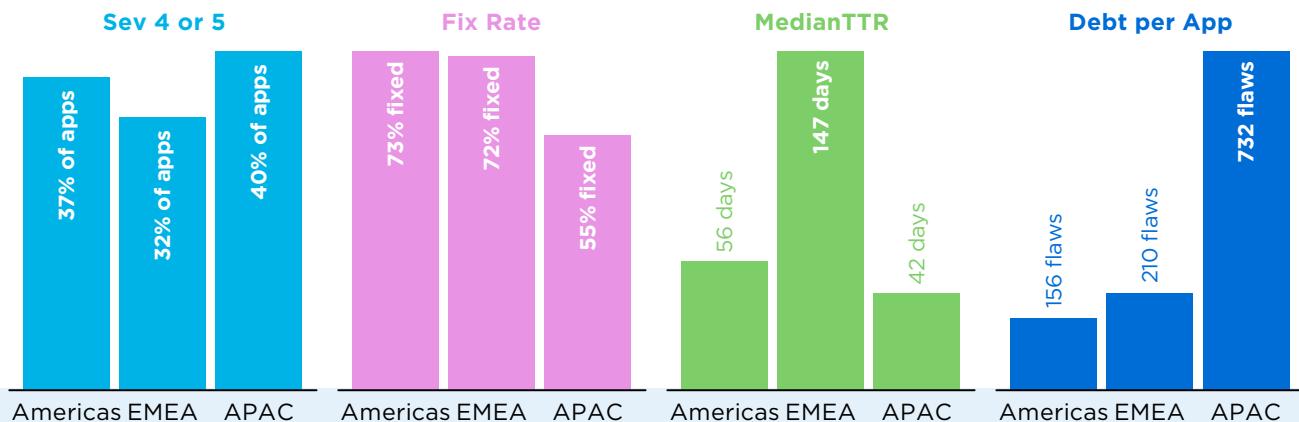


FIGURE 40

Comparison of key software security testing metrics by region

Source: Veracode SOSS Vol. 10

Statistics for the Americas track closely with the overall results presented thus far in the report. That's not terribly surprising, given the fact that the Americas — specifically the U.S. — dominate the sample data. But the other regions do show some interesting variation.

Beginning on the left, EMEA boasts the lowest prevalence of high-severity flaws, while APAC has the highest. The Americas and EMEA stand neck and neck in terms of fixing those flaws. Fix rates among organizations in APAC, however, are substantially lower. Even though APAC claims the quickest median remediation timeframes, it's not enough to offset the comparatively large amount of security debt accumulated among firms in that region. Organizations in EMEA generally appear to take longer to fix flaws, but still manage to keep debt under control — likely tracing back to the lower starting point for flaw prevalence.

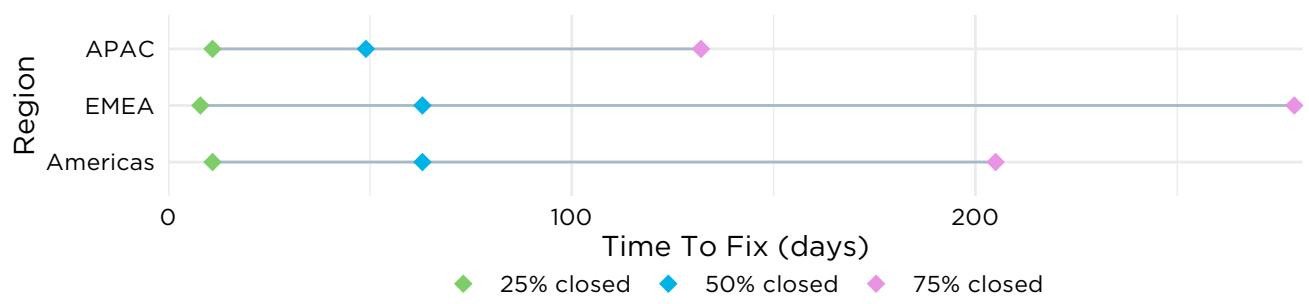


FIGURE 41 Comparison of time-to-remediation statistics across regions

Source: Veracode SOSS Vol. 10

Due to the substantial difference in MedianTTR for EMEA, we include Figure 41 to add more context. The major difference from the chart above is that this includes all flaws rather than just those rated as higher severity. In this expanded scope, EMEA's MedianTTR (50% closed) seems much more inline with other regions, though the long tail of slower fix timeframes remains apparent. APAC closes 75% of flaws in about half the time (yet still carries the highest amount of debt).

8

Key Takeaways

Any honest “state of” review is destined to be filled with a range of developments that can be considered encouraging, discouraging, and even uncertain. Our latest State of Software Security is no exception to the rule.

Part of the challenge in interpreting what the data says about the current state is that we're measuring a moving target.

Applications are living codebases that evolve over time in response to customer needs. Any change has the potential to introduce flaws that expose the application and organization to risk. Rather than fearing change, thereby conceding irrelevancy, development teams can embrace change by ensuring their ability to find and fix flaws keeps pace with their ever-evolving codebase. Our key takeaways reflect the evolving and challenging nature of this process.

#1

RECAP

Most applications have flaws, and it's critical that development teams find them before they're rolled into production.

The overall prevalence of flaws rose 11% since we first reported it 10 years ago, but the proportion of those flaws assessed to be of high severity dropped 14% over the same period. Many (us included) would view that as a net positive.

BOTTOM LINE

Prioritize flaws for efficient fixing.

With so many applications and flaws, it's easy to get overwhelmed. Don't assume recently discovered flaws are the most important. Even older Injection and Authentication flaws (which topped those used by exploits and incidents) can cause major problems.

#2

RECAP

The majority of flaws get fixed, but the time typically required to fix those flaws reveals a decade of no change (59 day average in 2010; 59 day median in 2019).

That seems discouraging at first, but considering the rapid proliferation of applications and vulnerabilities during that time, one could reasonably argue that developers are nobly standing ground in the face of overwhelming odds. We like this interpretation and are glad to support that mission.

BOTTOM LINE

Build habits around security activities.

Whether this is scanning codebases after every nightly build or remembering to follow a security checklist for all new features, creating a habit can be extremely useful. If you can find a corresponding trigger and reward for developers that encourage desired behaviors, you're most of the way toward creating a habit.

#3

RECAP

Security debt – defined as aging and accumulating flaws in software – is a challenge for all development teams.

About half of applications are accruing debt over time, a quarter driving it down, and another quarter breaking even. Our data offers strong evidence that a DevSecOps approach incorporating frequent, regular application testing cuts typical fix timeframes by 72% and decreases overall security debt by a factor of 5X!

BOTTOM LINE

Make a plan to pay down security debt.

Security debt can be thought of as analogous to personal credit card debt. If you spend every month and never make a payment, you'll have a whopping big bill that balloons over time. If you start paying for each month's new spending only, you'll never eliminate the balance (in fact, it'll keep growing due to interest). To get rid of the debt, you must address both the new spending and the balance.

So it goes with security debt.

Security findings are just like any other bug — developers create them along with writing new features, and there are usually a bunch more lurking in the code written years ago. You can choose to handle this one of three ways:

1

Ignore all the findings. Bad idea.

Like ignoring new charges on credit cards, eventually this leads to bad outcomes and further indebtedness.

2

Fix the new findings, ignore the old ones.

Also a bad idea. Leaving old flaws may be attractive to development teams because they probably didn't create them, but those long-unaddressed findings will inevitably come back to haunt you.

3

Fix the new findings and burn down the old in sprints.

Ideally, all of that happens every sprint, facilitated by frequent, regular security scanning (that's what the data would recommend). But special periodic "security sprints" could be run separately from normal finding/fixing to burn down security debt, provided you can accept the risk that those unresolved flaws may be exploited.

Ideally, you want to pay off ALL the debt as soon as possible, and then fix new findings every time they appear. This is like paying off your credit card in full every month — never living beyond your means so to speak.

Appendix

Methodology

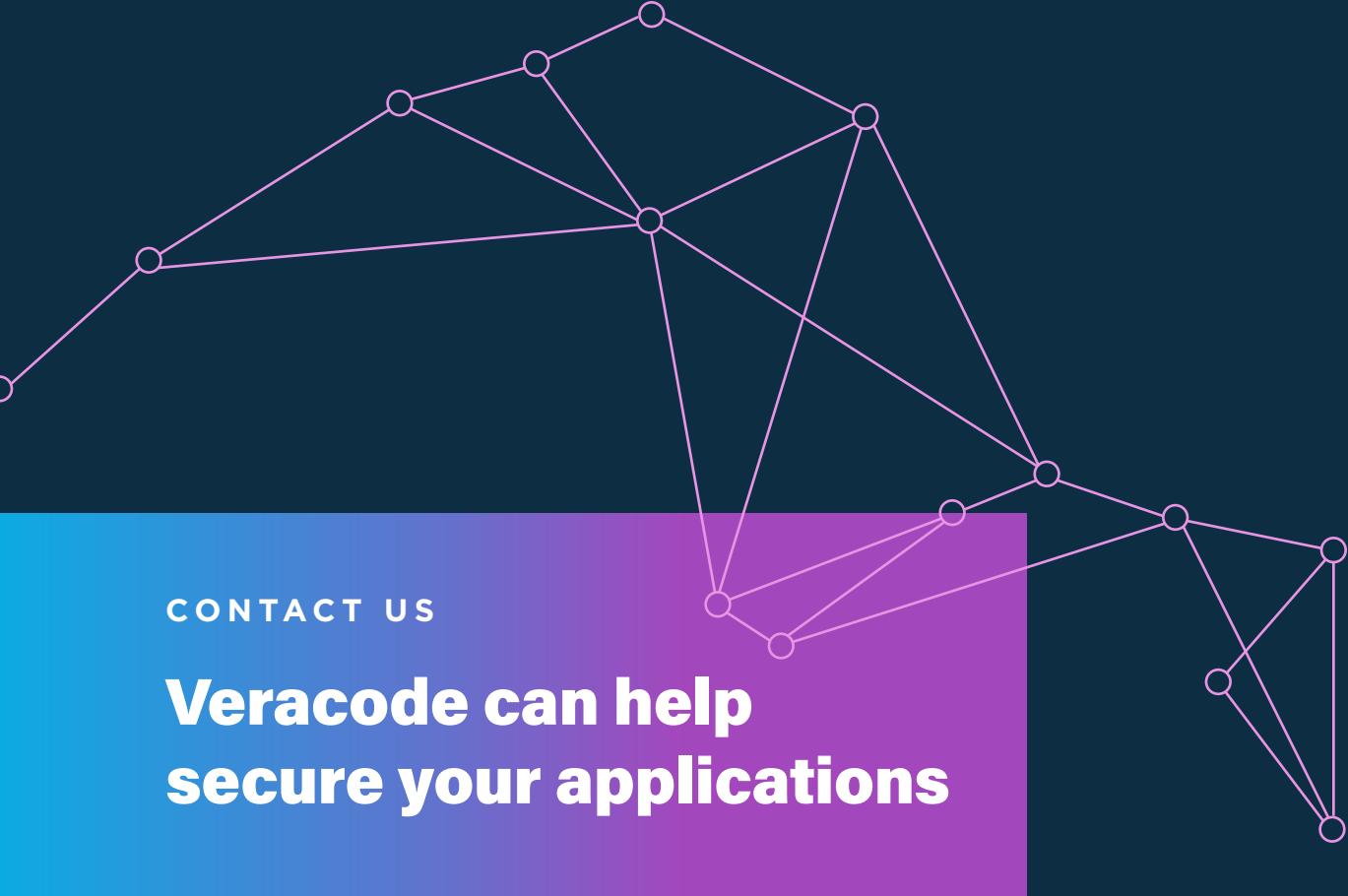
Veracode methodology for data analysis uses statistics from a 12-month sample window.

The data represents application assessments submitted for analysis from April 1, 2018 through March 31, 2019. The time-to-fix data stretches back farther than that because flaws closed in the sample period may have been opened well before it. The data represents large and small companies, commercial software suppliers, open source projects, and software outsourcers. In most analyses, an application was counted only once, even if it was submitted multiple times as vulnerabilities were remediated and new versions uploaded.

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 Level – predefined security policies which are based on the NIST definitions of assurance levels).

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% of scans closed three or less findings, 75% closed less than 8), 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: developers would scan entire filesystems, invalid branches or previous branches, and when they would rescan on the valid code, every finding not found again would be marked as "fixed." These mistakes had a large effect: the top one-tenth of one-percent of the scans (0.1%) accounted for almost a quarter of all the closed findings. These "mass closure" events have significant effects on exploring flaw persistence and time-to-remediation and were ultimately excluded from the analysis.



VERACODE

Veracode is a leader in helping organizations secure the software that powers their world. Veracode's SaaS platform and integrated solutions help security teams and software developers find and fix security-related defects at all points in the software development lifecycle, before they can be exploited by hackers. Our complete set of offerings helps customers reduce the risk of data breaches, increase the speed of secure software delivery, meet compliance requirements, and cost effectively secure their software assets — whether that's software they make, buy, or sell.

Veracode serves more than 1,400 customers across a wide range of industries, including nearly one-third of the Fortune 100, three of the top four U.S. commercial banks, and more than 20 of Forbes' 100 Most Valuable Brands. Learn more at www.veracode.com, on the Veracode blog, on Twitter and in the Veracode Community.