2025 STATE OF SOFTWARE SECURITY

A NEW VIEW OF MATURITY

VERACODE

Contents

Opening letter

Our research drives our own software security measures, and this year, in our 15th volume of this report, we seek to discover trends about where the most risk resides and what metrics can be used to gauge progress against it. Plus, we want to compare program performance of leading and lagging organizations using these metrics. The gaps between the top 25% and bottom 25% are fascinating.

Ultimately, realizing progress and maturity in software security requires a risk-based perspective. It takes focusing on the downside risks that matter in your context and the actions that create continuous feedback loops to see and remediate risk in an ongoing fashion.

This is easier said than done, so we hope you find the insights and guidance in this report as helpful as we have for improving security posture by adaptively securing mission-critical software in the artificial intelligence (AI) era.

Sincerely,

Niels Tanis
Senior Principal Security Researcher

Sohail Iqbal
Chief Information Security Officer

Chris Wysopal
Chief Security Evangelist

Executive Summary

In 2025, organizations face increasing threats to their software. The exploitation of vulnerabilities as the critical path to initiate a breach "almost tripled (180% increase) in the last year," according to the Verizon 2024 Data Breach Investigations Report.

Meanwhile, security debt is rising, and the attack surface is getting increasingly complex. Plus, the rise of AI in software engineering, especially with code generators, is transforming the risk landscape. While many teams may not openly admit to using AI, other indicators of its presence and impact can be found.

We also can't ignore the trends in the regulatory space that are happening in the U.S. and the E.U. In the EU, the Cyber Resilience Act went into effect December 2024 and focuses especially on enhancing the security of software. In the U.S. 2020 Biden Cybersecurity Executive Order emphasized cybersecurity prevention with Zero Trust network architectures and Secure by Design software. Secure by Design included static code analysis, dynamic code analysis, and supply chain security with SBOMs.

The U.S. Federal Government even required vendors to attest to the way they developed software as part of the acquisition process. Understanding your software risk posture is now a requirement. 2024 also gave us a new U.S. Securities and Exchange Commission (SEC) ruling which forces a more disciplined approach to cybersecurity risk management.

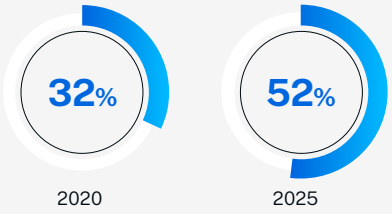
We believe these regulatory factors have contributed to some of the positive trends we see in the data, such as the OWASP Top 10 pass rate improving from 32% to 52% in the last five years.

However, our findings reveal that relying on traditional patching alone isn't enough. Security teams must take a more strategic, context-driven approach to managing the most urgent and exploitable risks. This requires seeing all risks in one place and focusing on what matters most to an organization.

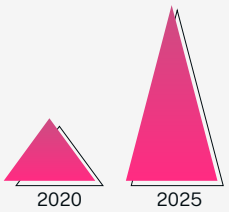
By prioritizing the most impactful risk remediation actions and creating continuous feedback loops for ongoing improvement, organizations can more effectively manage security risks over time.

Key findings

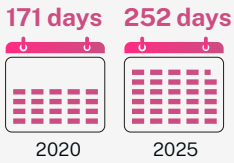
Good news first, the percentage of apps **passing the OWASP Top 10 has increased 63%** in 5 years (from 32% to 52%)



Now the bad news... the percentage of apps with **high severity flaws** has **increased by 181%**...

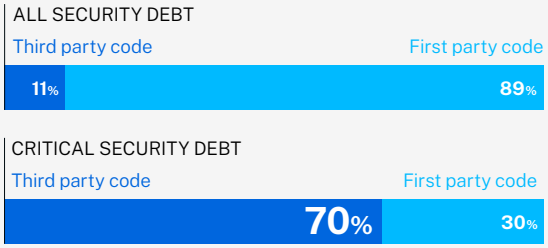


...and the average **number of days to fix flaws** has increased **47%**.



Half of organizations have **critical security debt** (high severity, high exploitability)...

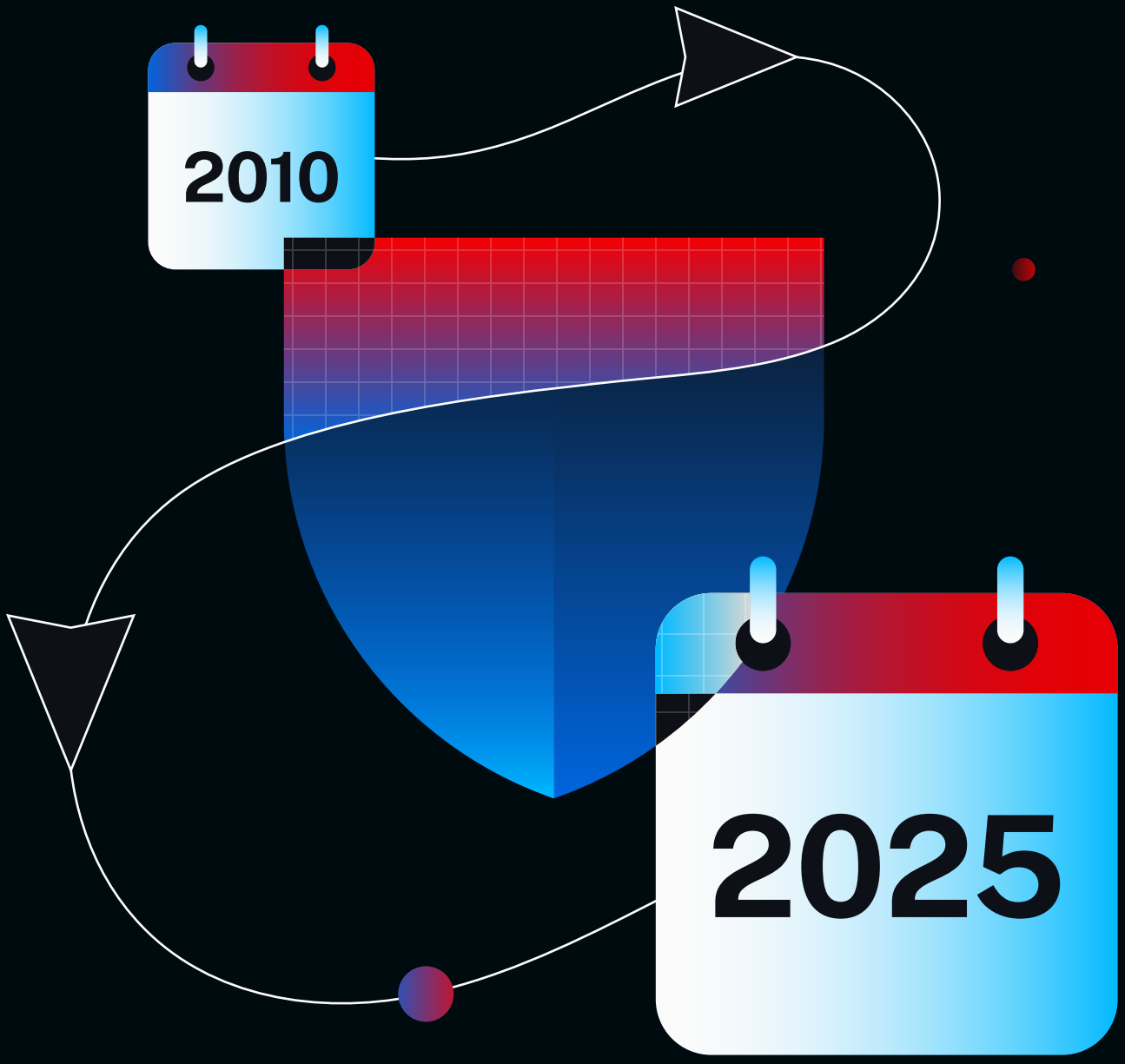
...and **70%** of it comes from **third party code** and the software supply chain.



The following table is a comparison of the top 25% and bottom 25% of organizations against 5 key metrics we've observed indicate the maturity of an organization at finding and fixing flaws in a way that systematically drives down risk.

	LEADING ORGANIZATIONS	LAGGING ORGANIZATIONS
FLAW PREVALENCE	**Below 43%**	**86% or more**
FIX CAPACITY	**Above 10% of flaws monthly**	**<1% of flaws monthly**
FIX SPEED	**Half of flaws in 5 weeks**	**Half of flaws in over a year**
SECURITY DEBT	**<17% of apps**	**>67% of apps**
OPEN-SOURCE CRITICAL DEBT	**<15%**	**100%**

15 Years of Special SoSS

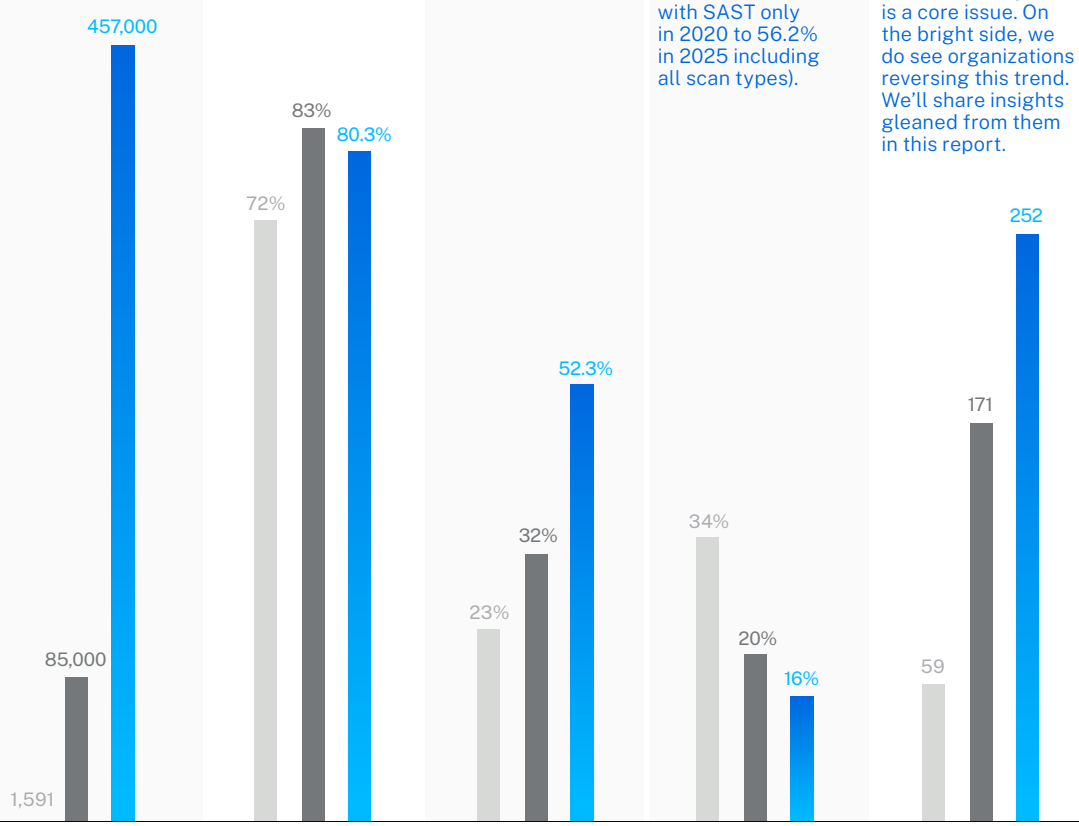


As a pioneer of the AppSec space, we have years of data to our advantage. This 2025 edition of the State of Software Security (SoSS) report is our 15th volume. That makes it a bit more special than the norm and creates an opportunity to highlight a few long-term trends before we dive into the latest facts and figures.

15 Years of Special SoSS

Volume 1 Volume 10 Volume 15

	NUMBER OF APPLICATIONS TESTED	APPS WITH AT LEAST ONE FLAW	OWASP TOP 10 PASS RATE	APPS WITH HIGH-SEVERITY FLAWS	AVERAGE NUMBER OF DAYS TO FIX FLAWS
⊕ Positive change since Vol 1 ⊖ Negative change since Vol 1	⊕ **+455,409**	⊖ **+11.5%**	⊕ **+127.4%**	⊕ **-52.9%**	⊖ **+193 DAYS**
	The sample size for this study has grown from ~1,600 applications tested in 2009 to nearly half a million in 2024! That strengthens the relevance of the findings in this report.	The fundamental challenge hasn't changed over the years: security flaws are very common across applications. Even so, there are signs of progress in software security.	The pass rate for scans of OWASP's most critical risks has more than doubled since Vol. 1. That means less risk for us all!	The prevalence of severe flaws in SAST scans was cut in half since Vol. 1. However, when you add in SCA (which only started in the last 5 years) and DAST, the increase in high-severity flaws is 181% since 2020 (from 20% with SAST only in 2020 to 56.2% in 2025 including all scan types).	One aspect of AppSec that's gotten worse over iterations of the SoSS is the time it takes to fix flaws. There are many reasons for this, but the ever-growing scope and complexity of the software ecosystem is a core issue. On the bright side, we do see organizations reversing this trend. We'll share insights gleaned from them in this report.



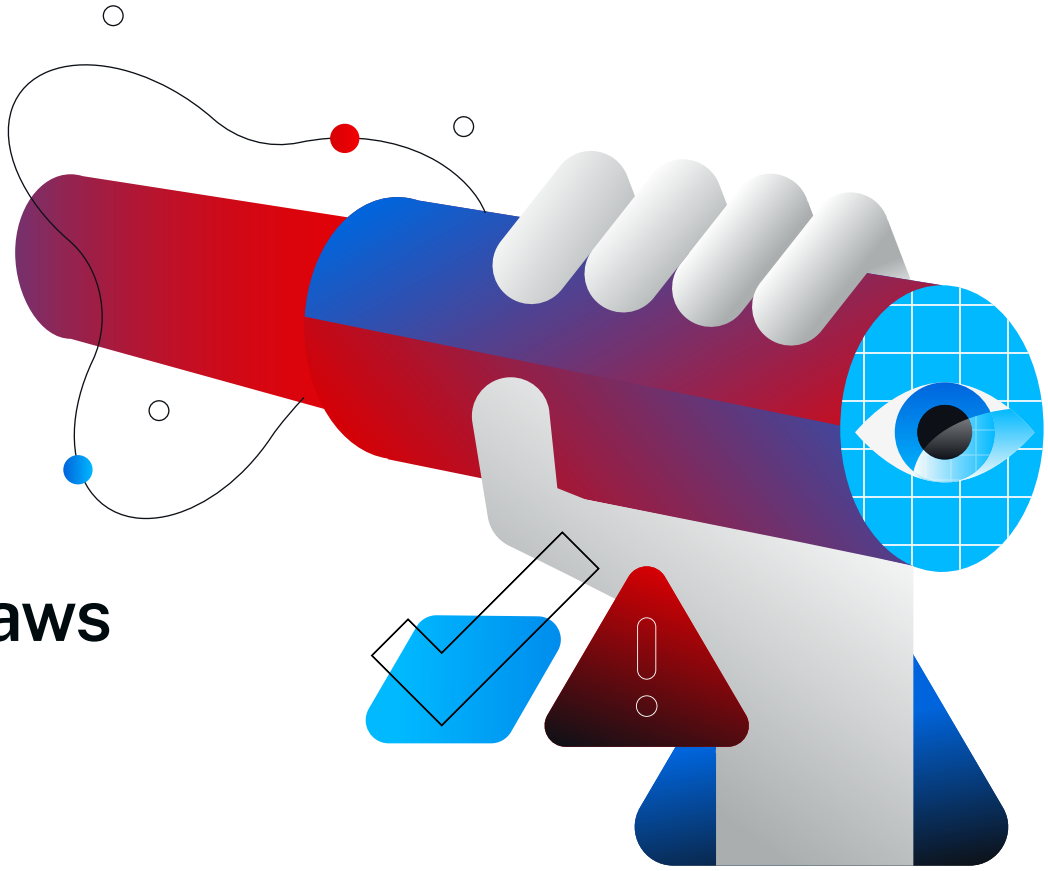
1. All statistics in this 15-year retrospective are based on static analysis (SAST) scans only because that's consistent with early versions of the SoSS. You'll see that the "State of" section shows some very different results from the latest data drawn from all SAST, dynamic analysis (DAST), and software composition analysis (SCA) scans (e.g., 56% of apps have high-severity flaws). Combined stats from all scan types is the norm for this report unless otherwise noted.

State of Software Security in 2025

At this point, everyone even remotely associated with software security is familiar with phrases like “We need to shift left” and “Secure your supply chain.” Those are worthy aspirations to be sure, but what, exactly, do they entail, and where are we along the road to getting there?

In a nutshell, shifting left and securing software supply chains involves finding and fixing security flaws before they get rolled into production applications that place organizations at risk. That process of finding and fixing flaws—and fighting the security debt that results from not fixing them fast enough—happens to be something into which we have a unique vantage point. And we’re glad to have the opportunity to once again share what we’ve observed about the state of software security over the last year. Let’s get started!

The findings analyzed in this report were discovered via 1.8 million SAST, DAST, and SCA scans of nearly half a million applications. In most cases, we show combined results from all three types of tests (e.g., Figure 1) but occasionally feature one of them (e.g., SAST only in Figure 2). We’ve designated charts based on specific scan types in the captions.



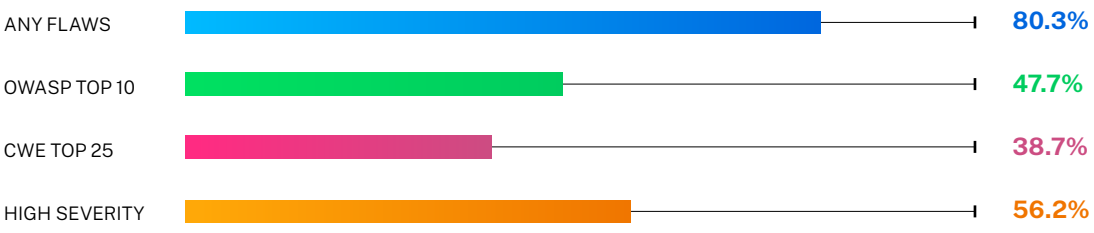
Finding flaws

Any substantial codebase has bugs at some point in its lifecycle. Some of those bugs undermine the confidentiality, integrity, or availability of the application, thereby placing the organization at risk. Figure 1 reveals that 80% of the applications tested over the last year have at least one security flaw.

Figure 1 includes how we categorize flaws. Just under half of all applications have flaws ranked in the OWASP Top 10 as the 10 most critical risks and over one-third contain those considered most dangerous, per the CWE Top 25. Over half exhibit high or critical severity flaws according to our own rating system.

FIGURE 1

Percent of applications with security flaws

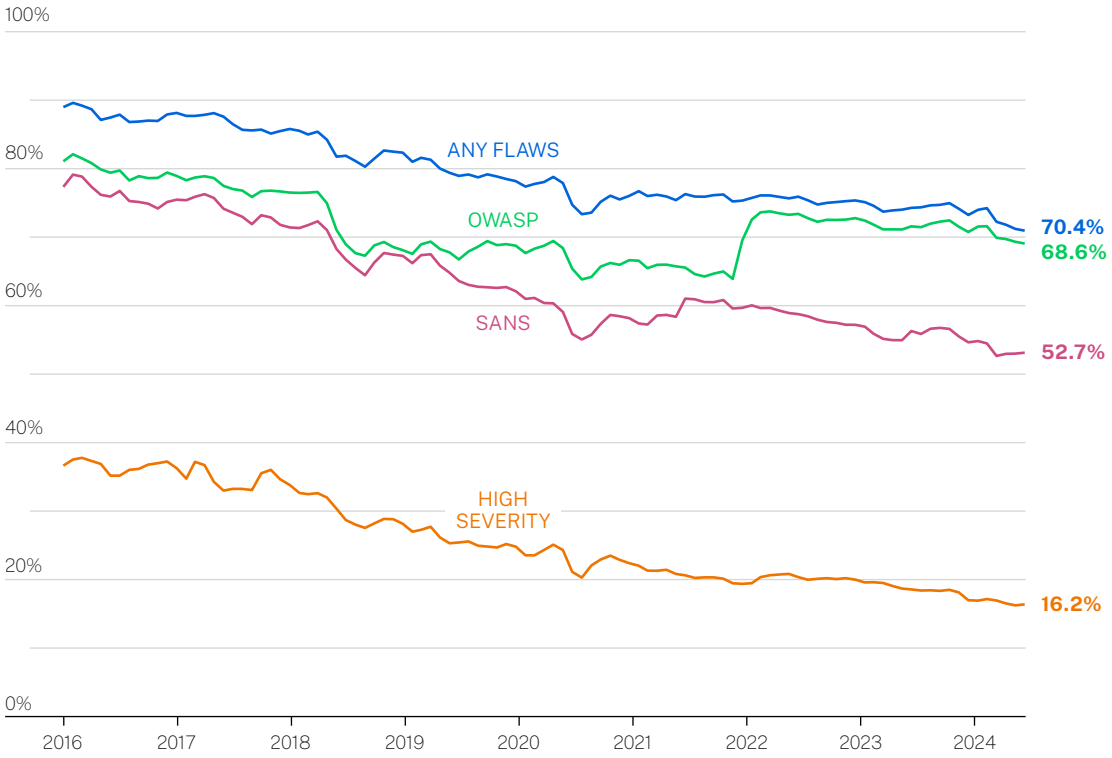


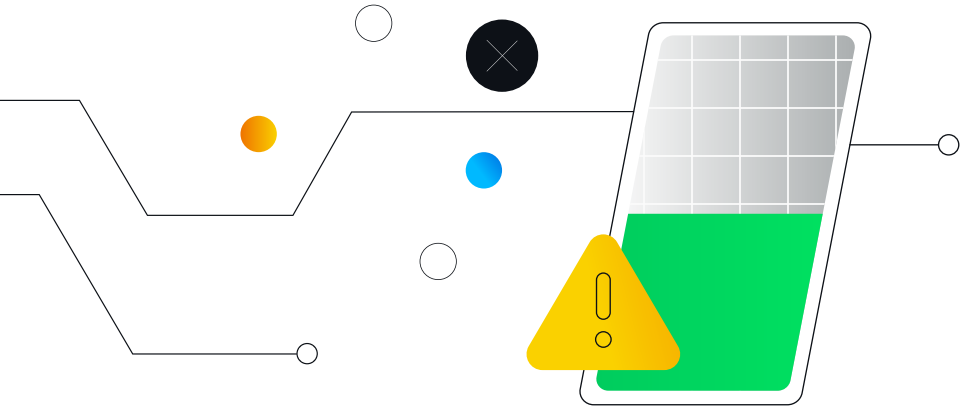
While the level of flaws, and specifically high severity flaws, remains high, we're happy to report that the proportion of applications failing OWASP Top 10 and CWE Top 25 tests is steadily declining. Of particular note, the prevalence of high-severity flaws has been cut in half over the last decade.

FIGURE 2

Prevalence of security flaws over time (SAST only[2])

Percent of applications





The prevalence of high-severity flaws has been cut in half over the last decade

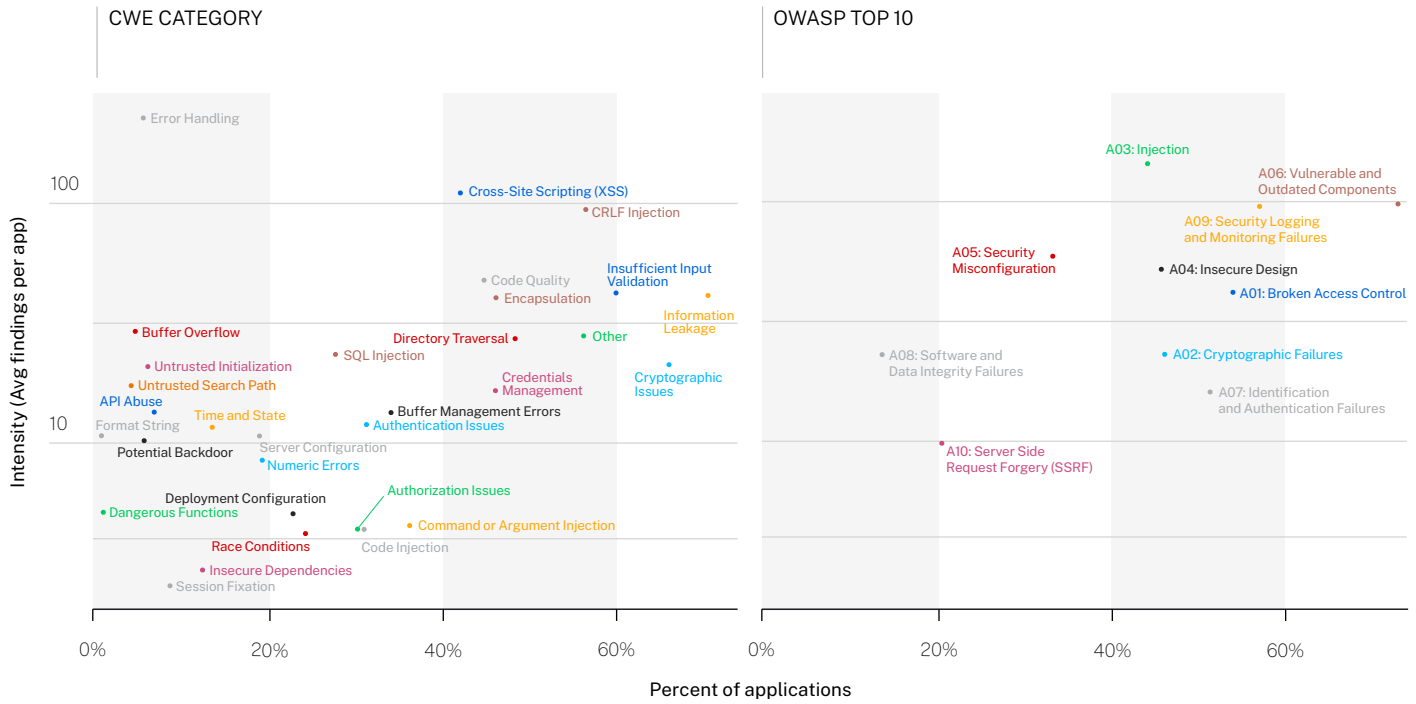
2. Remember that findings in this report combine SAST, DAST, and SCA scans unless otherwise noted, as we've done here. Figures 1 and 2 differ so much because the latter is based only on SAST scans.

The stats shared thus far are for any type of flaw. That's a good "thumb on the pulse" indicator of the state of affairs in software security, but let's dig a little deeper. Figure 3 starts by digging into the types of flaws detected over the last year for CWE and OWASP categorizations.

Any flaw categories toward the right affect large numbers of applications, and those near the top occur frequently within the codebase of those applications. Take note of those in the upper-right danger zone, as they represent the most common security bugs crawling around your code.

FIGURE 3

Prevalence and intensity of CWE and OWASP flaws in applications



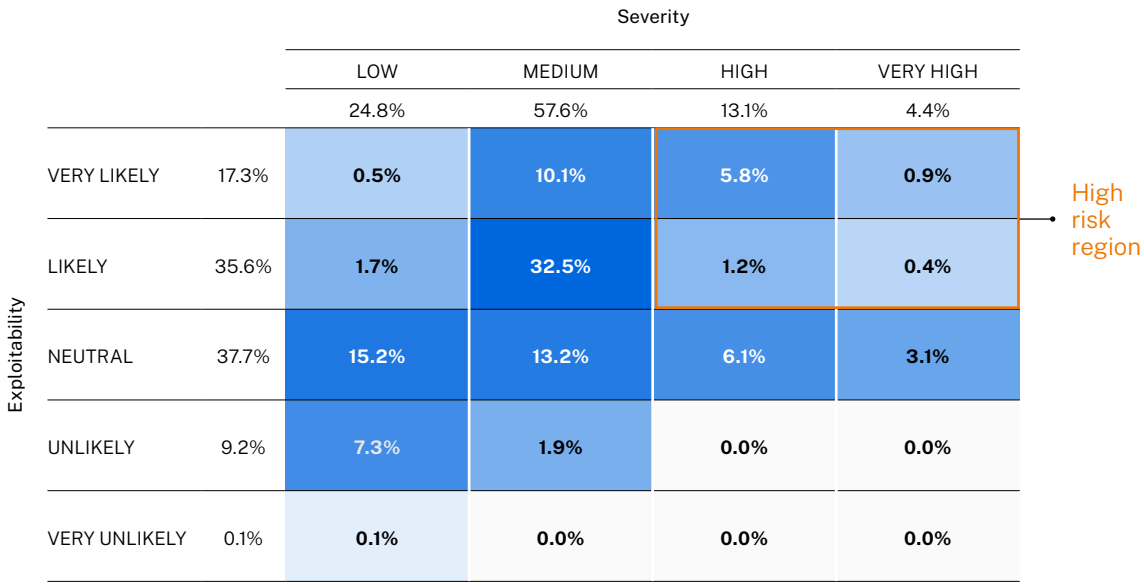
Looking for more info on OWASP Top 10 flaws, how Veracode tests for them, and what you can do to prevent them? Find out more on our Knowledge Base article.

In our analysis, we use two ways of assessing the overall risk posed by security flaws: severity and exploitability. The former reflects the potential impact on confidentiality, integrity, and availability, and the latter rates the likelihood or ease with which an attacker could exploit a flaw. Figure 4 gives a breakdown of all flaws according to these ratings.

Only a small minority of flaws (8.4%) rank high for both severity and exploitability. We'll soon see that many organizations struggle to fix flaws in a timely manner, so it's all the more important to prioritize those that represent the highest risk. The "High-Risk Region" in the upper-right corner of Figure 4 is a great place to focus remediation efforts.

FIGURE 4

Breakdown of flaws according to severity and exploitability



Pretend for a moment that your team began writing flawless code—would that spell the end of security issues plaguing your applications? Unfortunately not, because your applications include a plethora of open-source libraries written by third parties that don't share your newfound ability to code perfectly. About 7 in 10 applications tested by Veracode contain flaws in third-party code. For those keeping score, that's 6% higher than the flaw prevalence for code written by your developers!

FIGURE 5

Prevalence of flaws in first-party vs. third-party code among applications

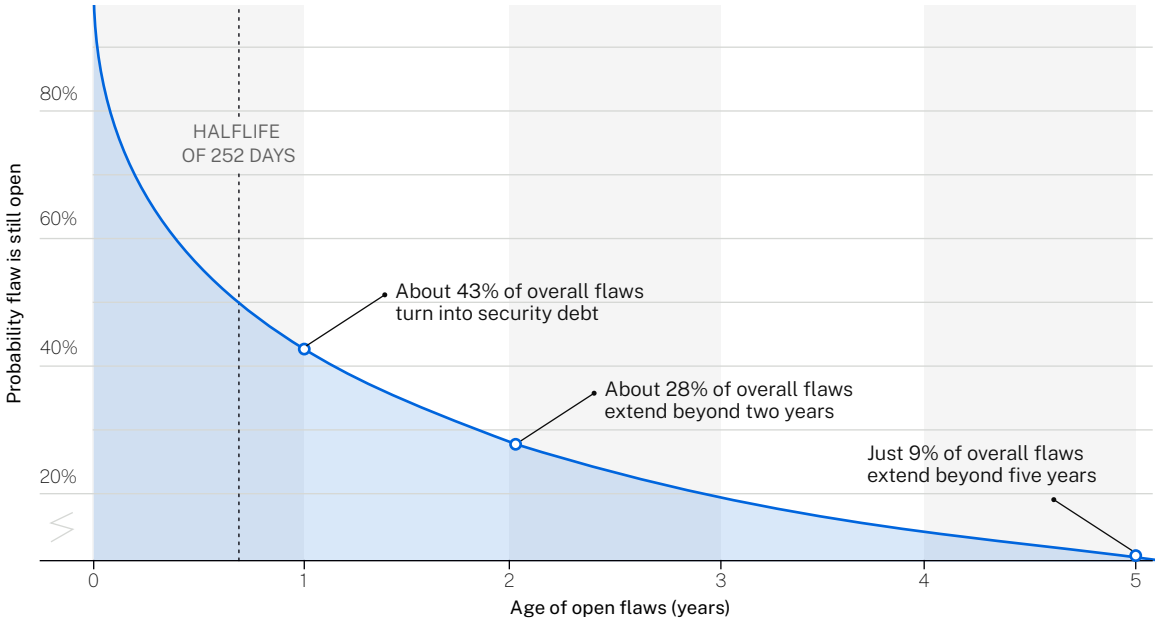
Fixing flaws

Finding flaws is easy these days; fixing them is where the challenge lies. That's why we've put a lot of focus over the years on helping your AppSec programs mature in speed and efficacy of detection, prioritization, and remediation. There are many ways to measure this, with simple averages of flaw closures being the most common. But that approach a) ignores the persistence of unresolved flaws, and b) isn't appropriate for long-tailed distributions like we see for remediation timelines.

Survival analysis offers the most realistic depiction of flaw remediation timelines. A flaw's lifespan begins at discovery and ends when scans confirm that it has been fixed.[3] Figure 6 depicts the overall survival curve for all types of flaws across all applications. You can determine the survival rate at any point based on where the x and y axes intersect along the curve. For example, 28% of flaws are still open two years after being discovered. After five years, 9% of flaws linger on.

FIGURE 6

Overall flaw remediation timeline based on survival analysis



3. Another benefit of survival analysis is that it accounts for "censored data" that includes flaws still open (or, at least, not verified as closed) when our measurement period ends.

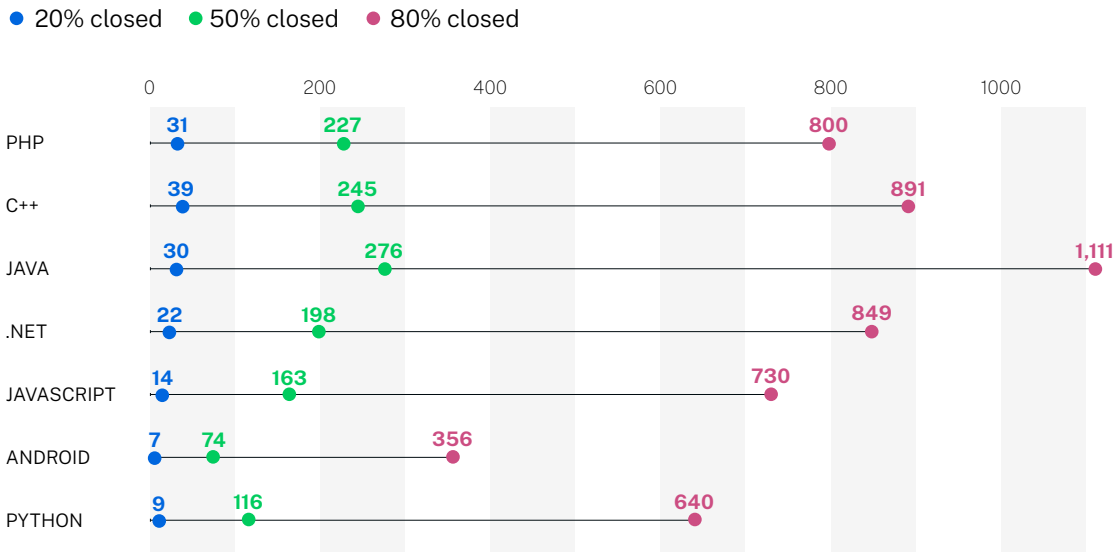
Half-life is a key statistic associated with survival analysis, measuring the time it typically takes to fix 50% of flaws. Overall, the half-life of flaws stands at just over eight months. As you may suspect, this statistic varies greatly among different flaws, applications, and teams. Take, for instance, the development languages in Figure 7. The half-life for flaws in Android apps is about one-fourth that of Java, and the 80% fixed threshold is crossed almost two years sooner!

Figure 7 is a simplified view of fix times based on the same survival analysis technique. The points mark the time to remediate 20%, 50% (half-life), and 80% of flaws in each category.

FIGURE 7

Flaw remediation timelines among application languages

Time to fix, days

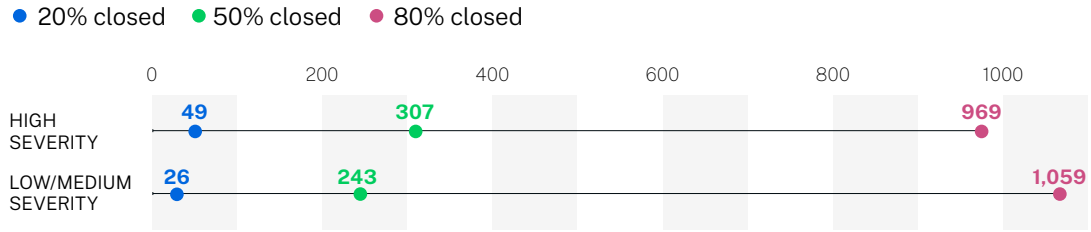


While we'd like to see the riskiest flaws fixed ASAP, the data suggests that severity isn't a major driver of remediation for most teams. The half-life of critical flaws is only about a month shorter than that of less severe findings. Circling back to the high-risk ratio in Figure 4, prioritizing fixes of critical flaws represents a huge opportunity for organizations to efficiently reduce their exposure.

FIGURE 8

Flaw remediation timelines for high-severity vs. lower-severity flaws

Time to fix, days



The shape of the survival curve in Figure 6 makes it clear that the process of fixing flaws begins in earnest but tapers off over time. The longer a flaw survives, the less likely it is to be resolved. There are numerous reasons for this phenomenon, but the result is that applications gradually become bloated with old, unresolved flaws, which we term security debt.

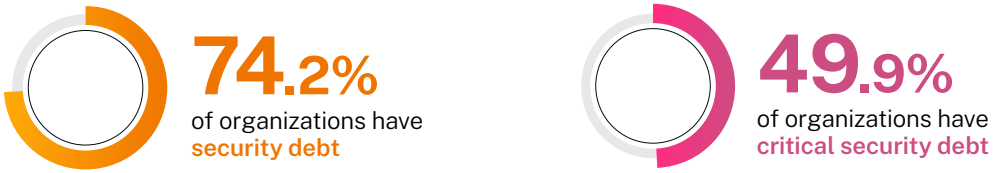
Fighting debt

Security debt, a term that's become common parlance in this report, refers to flaws that remain unfixed for over a year. How common is this problem? Almost three-quarters of organizations have accrued some level of debt, according to Figure 9.

Moreover, half of them exhibit critical debt—the risky combination of highly severe and long-unresolved flaws. The flip side of this statistic is that a quarter of organizations manage to stay out of debt completely. Kudos to them. Again, those wins need to be recognized.

FIGURE 9

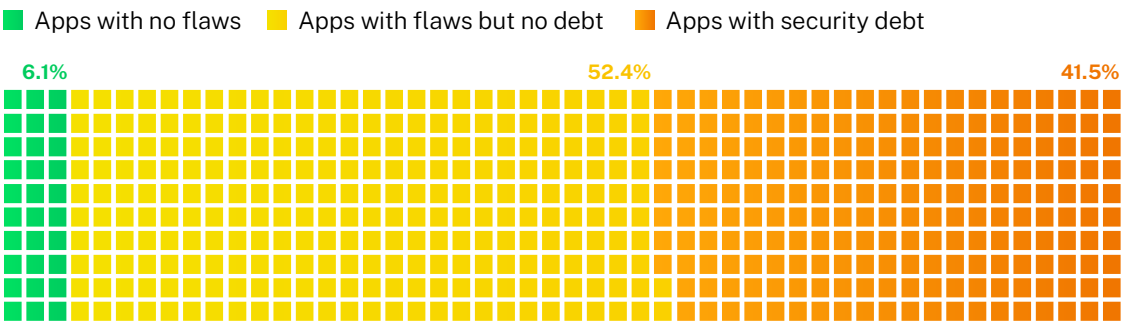
Prevalence of security debt and critical debt among organizations



Security debt is also prevalent at the application level. We found flaws constituting security debt in about 42% of all actively tested applications,[4] which remains unchanged from our last report. Sure, we'd like to see the debt ratio start to fall, but at least it hasn't gotten worse. There's solid evidence in our data that organizations can drive down debt, and we've collected insights on how yours can accomplish that feat in the second half of this report.

FIGURE 10

Prevalence of security debt across all applications active for at least one year

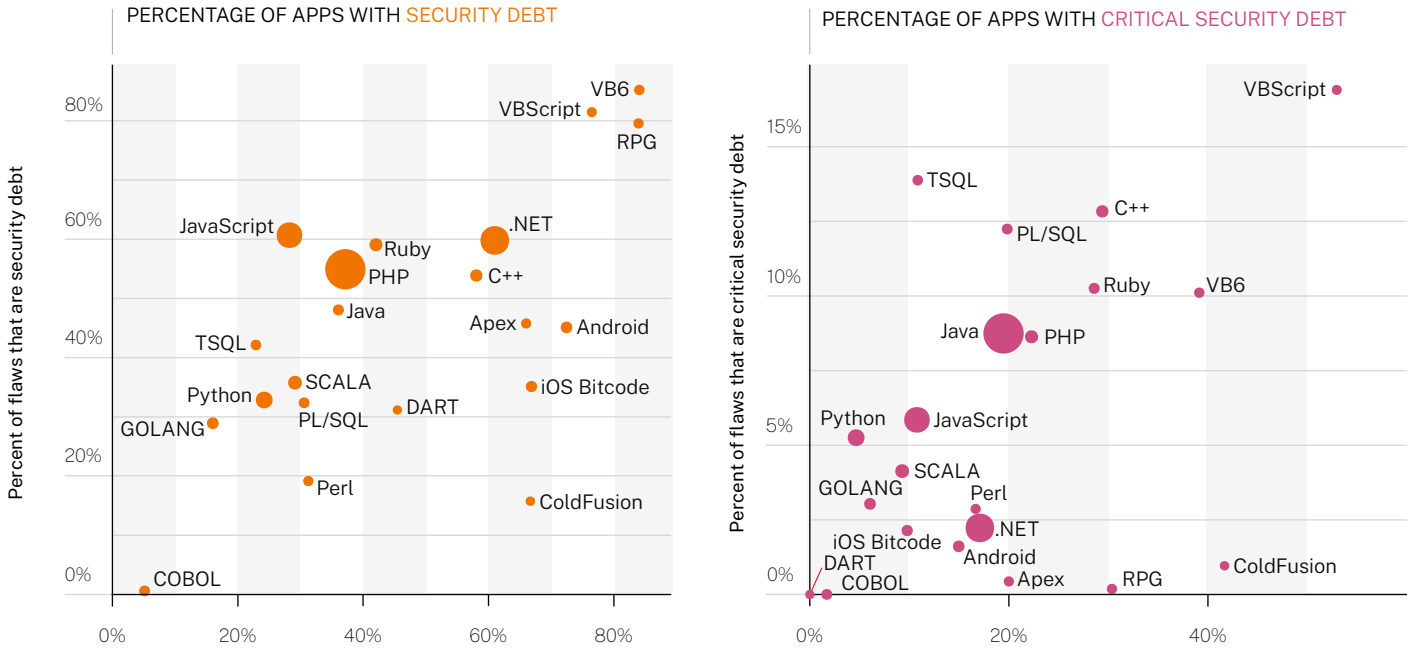


4. We filtered this to applications that have been actively tested for at least one year to allow for the accrual of debt. If we remove that filter, 21% of all tested applications have security debt.

It's clear that some coding languages are more predisposed to the accrual of security debt than others. For this reason, more mature organizations develop a language-based strategy to fight security debt. This can be seen in Figure 11, where VB6 and COBOL are polar opposites in terms of the prevalence of security debt. Granted, those aren't today's most popular languages, but more common ones like .NET, Java, and Python still show significant variation.

FIGURE 11

Prevalence of security debt by application development language

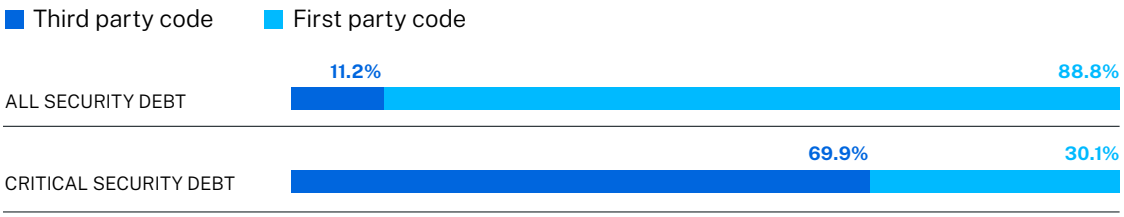


Not only does debt affect the code your developers write, but it also creeps in via open-source libraries imported into your applications. Of all security debt we detected in the last year, a relatively small percentage (11%) stemmed from third-party code. But if we look specifically at critical security debt, that ratio jumps to 70%. Any debt fighting strategy that doesn't extend beyond your team's own code is not one that will ultimately be successful.

FIGURE 12

Proportion of security debt and critical debt in first-party vs. third-party code

Percentage of flaws

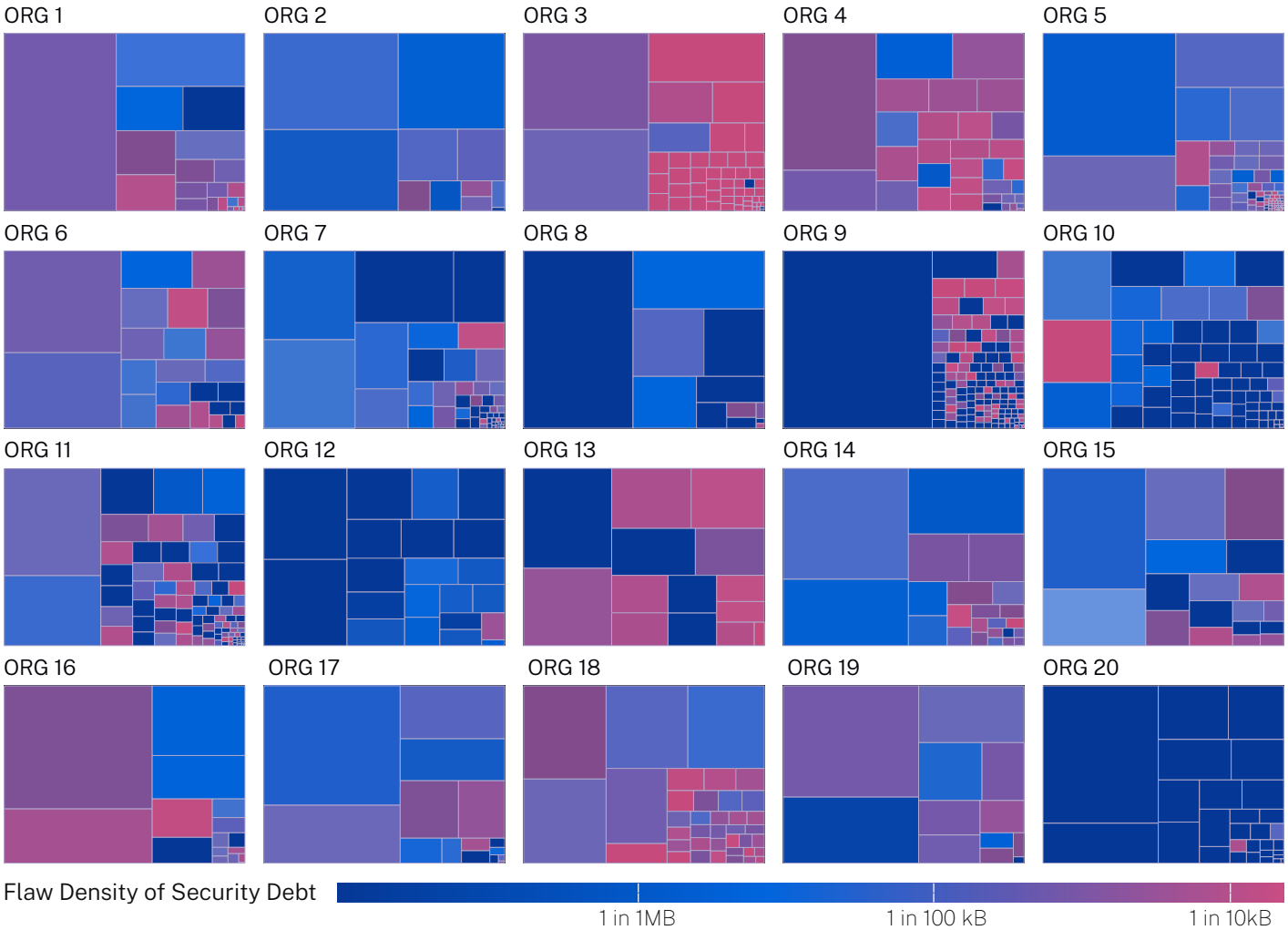


Comparing Software Security Program Performance

One of the charts from last year's SOSS that resonated with readers highlighted the disparity among organizations in managing security debt. We've reproduced that chart below using the latest data. Each box represents an anonymous organization, and the internal rectangles correspond to their active applications of differing sizes. The color applied to those applications measures the density of security debt (red indicates higher density).

FIGURE 13

Distribution of security debt across applications in 20 example organizations



As illustrated, some organizations have almost no security debt (Org 20), while others are drowning in it (Org 3). Most fall somewhere in between, with a mix of debt-free and debt-ridden applications. These results raise the question of what factors account for the marked differences in how these organizations manage security debt. Or, more to the point—what can your team(s) do to achieve results that look more like Org 20 than Org 3?

Answering that question begins with assessing where your organization stands with respect to factors that contribute to security debt. This section supports that assessment with five key software security metrics. We define each metric, explain its importance, benchmark performance, and share recommendations from leading organizations and our experts.

Flaw prevalance

What is it?

Flaw prevalence measures the percentage of applications with at least one unresolved security flaw in the latest scan or test. This can be calculated as an overall metric or for groups of applications, development teams, or types of flaws.

Why does it matter?

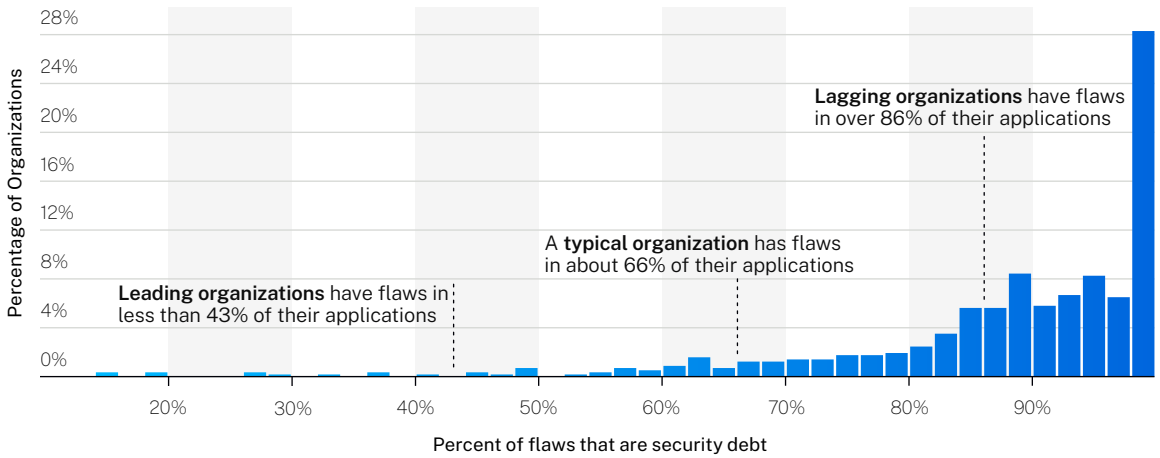
The value of this metric isn't in discovering that most of your applications have security flaws but rather in establishing a baseline that can be tracked over time. Significant changes—positive and negative—can be reviewed to discern what might have caused them, and those insights can then be used to improve secure development processes.

Where do we rank?

We see in Figure 14 that the prevalence of security flaws is quite high in most organizations. The annotations offer stats to dial in that general observation with comparable metrics. The typical organization has security flaws in about two-thirds of its applications (median of 66%). Leading organizations maintain a flaw prevalence below 43%, while lagging firms struggle with twice that proportion (86% or more).

FIGURE 14

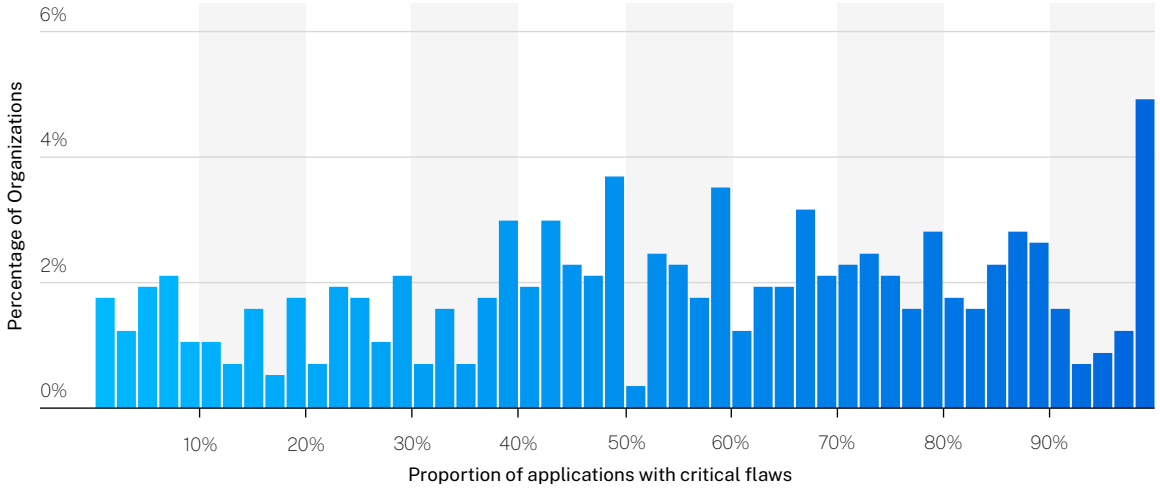
Overall flaw prevalence among organizations



With security bugs being fairly common across all firms, a strong case can be made that focusing on the riskiest flaws makes for a better KPI. We see in Figure 15 that the prevalence of high-severity security flaws is more evenly distributed, with a median of 50%. Top performers keep these risky flaws from affecting no more than 20% of their applications.

FIGURE 15

High-severity flaw prevalence among organizations



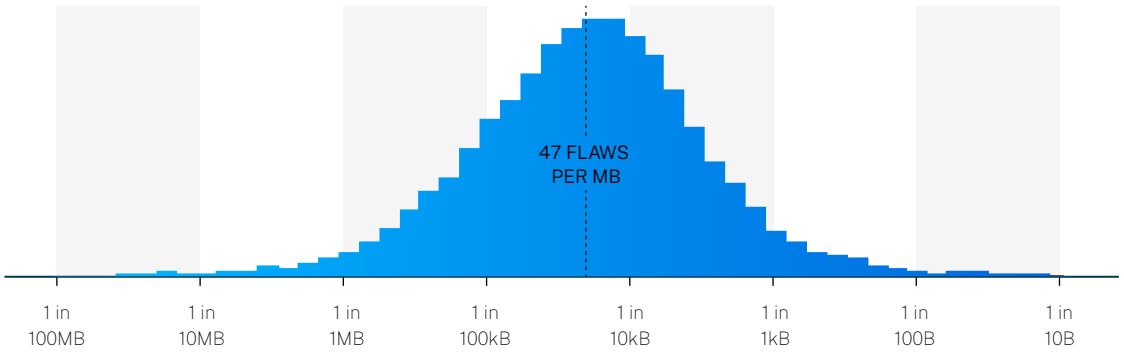
What else should I know?

If you'd like to measure the prevalence of flaws *within* applications in addition to across them, flaw density is your metric. Flaw density normalizes the number of flaws relative to the size of the application to aid comparisons. A typical application has about 47 flaws for every 1 MB, which, by itself, isn't a very meaningful statistic. Figure 16 shows the distribution to aid benchmarking. The flaw density of leading firms (20th percentile) is 19 times lower than lagging firms (80th percentile).

FIGURE 16

Density of flaws detected in applications

Flaw density per application



NOTE: A high flaw prevalence isn't necessarily a bad thing - especially for maturing programs. The more comprehensive your AppSec program becomes and the more types of scans you use, the more flaws you will find. This is a good thing. That said, you want to whittle down flaw prevalence over time through automations in the SDLC, and discover what's most severe, exploitable, and urgent to tackle first. More on this in the recommendations later.

Fix capacity

What is it?

Fix capacity calculates the number of security flaws remediated in a given timeframe as a percentage of all flaws detected for an application. We typically measure it on a monthly basis and then average those values over time.

Why does it matter?

It's clear from the prior section that the volume of security flaws can be overwhelming for many organizations. Fixing them all isn't feasible, or even necessary, for most teams (at least not immediately). But some teams, for whatever reason, can consistently fix more than others.

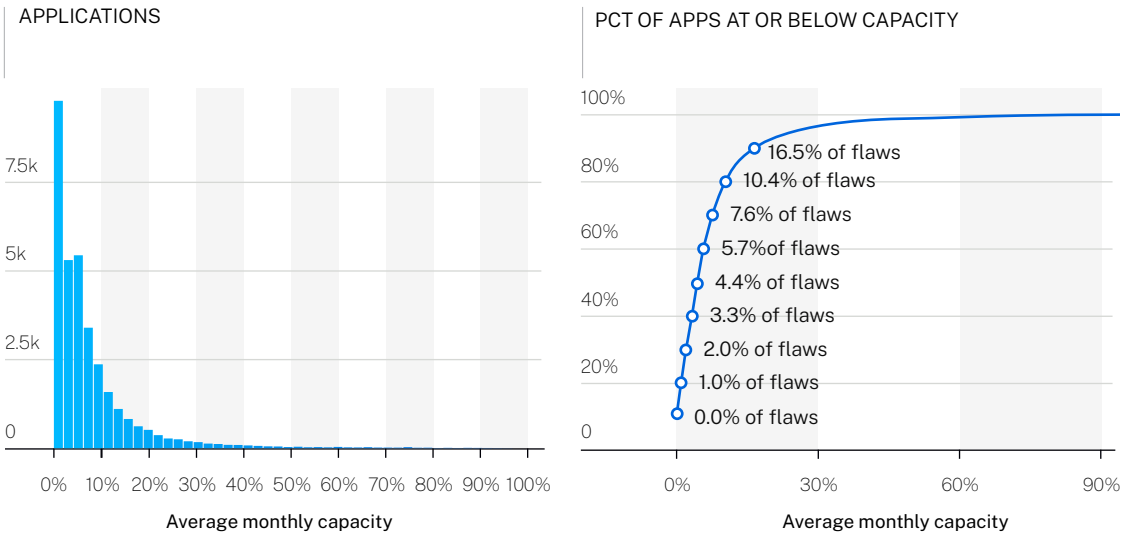
This metric helps teams assess what proportion of existing flaws they can reasonably expect to fix in the next month or quarter and plan accordingly.

Where do we rank?

Per the chart on the left, the average monthly fix capacity for most applications is less than 10% of all flaws. Some applications boast higher rates, but they drop off quickly. The chart on the right makes it easier to distinguish top and bottom performers. Leading teams have fix capacities above 10%, while the bottom tier fixes just 1% of its flaws each month.

FIGURE 17

Average monthly fix capacity across applications



What else should I know?

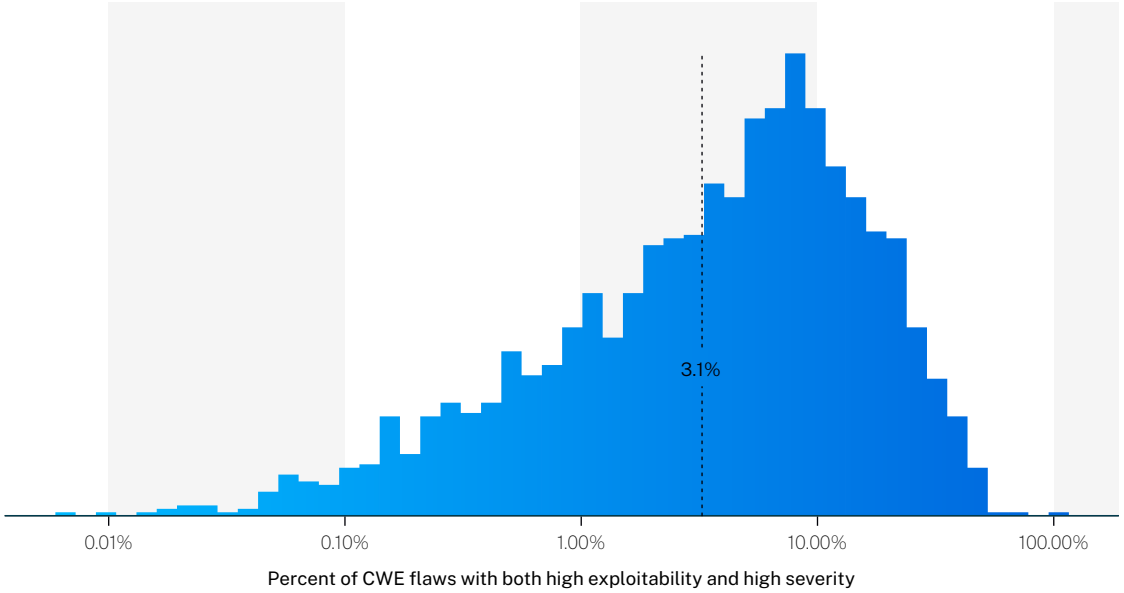
Any discussion of fix capacity prompts questions to the effect of, "Is it enough?" The low capacities revealed in Figure 17 make it clear the answer is "Not even close." Keep in mind that if you fix 10% of flaws per month, you can't expect to knock them all out in 10 months because new ones are added (and found) in the development process.

Now, let's rephrase the question to "Is it enough to fix the riskiest flaws?" To answer that, we first need to define what those are. The upper-right quadrant of Figure 4 shared earlier in this report is a good place to start. Just over 8% of all flaws are rated high for both exploitability AND severity.

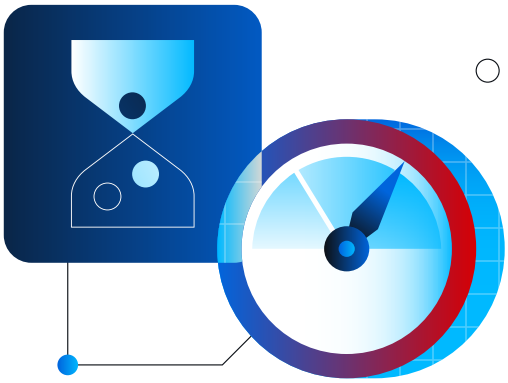
This critical risk ratio—which could very well be its own metric—ranges from 0.2% to about 11% for the majority of organizations. That's much more in line with the fix capacities observed in Figure 17, meaning this may be a good focal point for organizations trying to get the most bang for their flaw-fixing buck.

FIGURE 18

Critical flaw ratio among organizations



NOTE: Though the word "capacity" connotes an inherent limitation, it's more of a choice than a ceiling. For example, organizations choose how much effort goes into fixing flaws vs. adding features. They decide which flaws need to be fixed and which don't. They schedule deadlines for those fixes. All of these choices, and many more, affect capacity as measured here. There's another aspect of capacity which is increasing the efficiency of the time spent fixing. If you're measuring capacity as the number of hours developers spend fixing, then capacity can remain unchanged but the number of fixes happening within that time increases through flaws being fixed closer to where they were created or by utilizing AI to help with remediation.



Fix speed

What is it?

Fix speed measures the rate at which flaws are fixed across all active applications. As discussed previously, we use survival analysis as the basis for this metric. That said, we still need a specific point on the survival curve (see Figure 6) to use as the basis for comparisons. The half-life of security flaws serves this purpose well and is defined as the time it takes to fix 50% of all security flaws discovered.

Why does it matter?

A strong argument can be made that how flaws are handled once they're detected says more about an organization's approach to software security than how many flaws are introduced. Slow or sporadic remediation suggests a lack of urgency or capability (or both) on the part of the organization to reduce exposure before code is pushed into production applications.

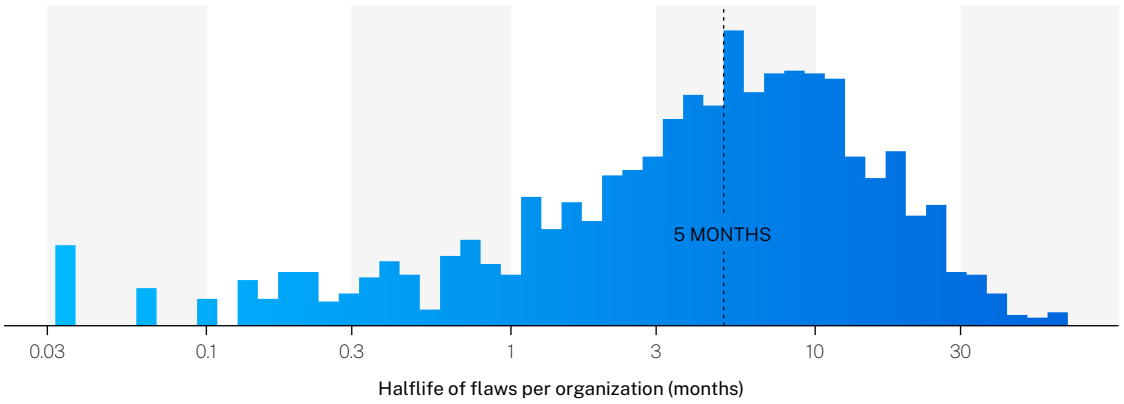
Where do we rank?

The typical organization takes about five months to fix half of all detected security flaws (that's the median). Leading teams cross the halfway point in roughly five weeks, while half-life spikes just above one year among lagging organizations. It's also worth noting that there are a few firms well outside that range, achieving half-lives as short as 1 day and as long as 3.5 years!

As shown earlier in this report, fix speed doesn't shift dramatically based on flaw severity. The median half-life of critical flaws is still quite high at 3.7 months. Even leading organizations only manage to move the needle by about two weeks (half-life of three weeks for critical flaws vs five weeks overall).

FIGURE 19

Fix speed (flaw half-life) among organizations



What else should I know?

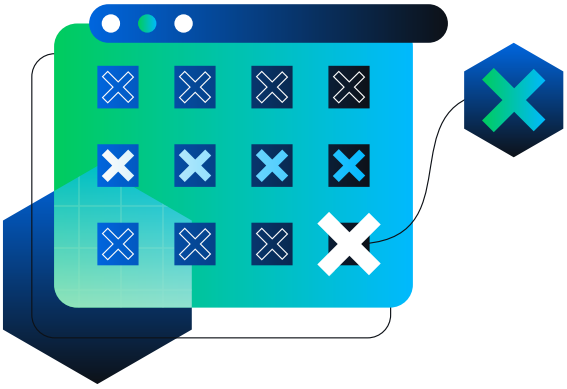
In prior editions of this report, we've identified several measurable factors that significantly improve flaw remediation timelines. Frequent testing of applications, using multiple types of tests (SAST + DAST + SCA), and security training for developers all correlate with faster fixes. And the latter appears to be increasingly effective.

When we first measured this back in 2021, we observed a two-month reduction in flaw half-life among organizations that made use of Veracode's Security Labs. That delta stood at about four months in the 2024 SOSS. The most recent data in Figure 20 reveals that teams using Security Labs boast flaw half-lives that are 7.5 months shorter than those not leveraging these resources!

FIGURE 20

Comparison of remediation timelines for teams that use vs. don't use Security Labs

NOTE: Thanks to DevOps, organizations are experiencing new levels of speed and flexibility when it comes to software development. But too often, speed comes at the cost of security. The faster teams need to deliver code, the easier it is to downplay security or put it off for later. By speeding past security in the development stage, however, organizations are often forced to exponentially slow down later when the product goes to production and security flaws are discovered and must be fixed.



Debt prevalence

What is it?

Security debt refers to security flaws that persist for at least one year after discovery. Debt prevalence measures the percentage of applications that have accrued unresolved flaws exceeding this threshold. All security debt is bad, but some are worse than others. Thus, we distinguish critical debt as a particularly concerning class of debt consisting of persistent high-severity flaws.

Why does it matter?

This metric helps assess whether security debt is widespread across your applications or limited to a small subset. A low value suggests isolated or temporary issues, while a higher debt ratio points to more pervasive and persistent problems. Like massive debt of a financial nature, the latter state is much more challenging to address. Driving down and eventually eliminating security debt—especially critical debt—should be a top priority for development and security teams.

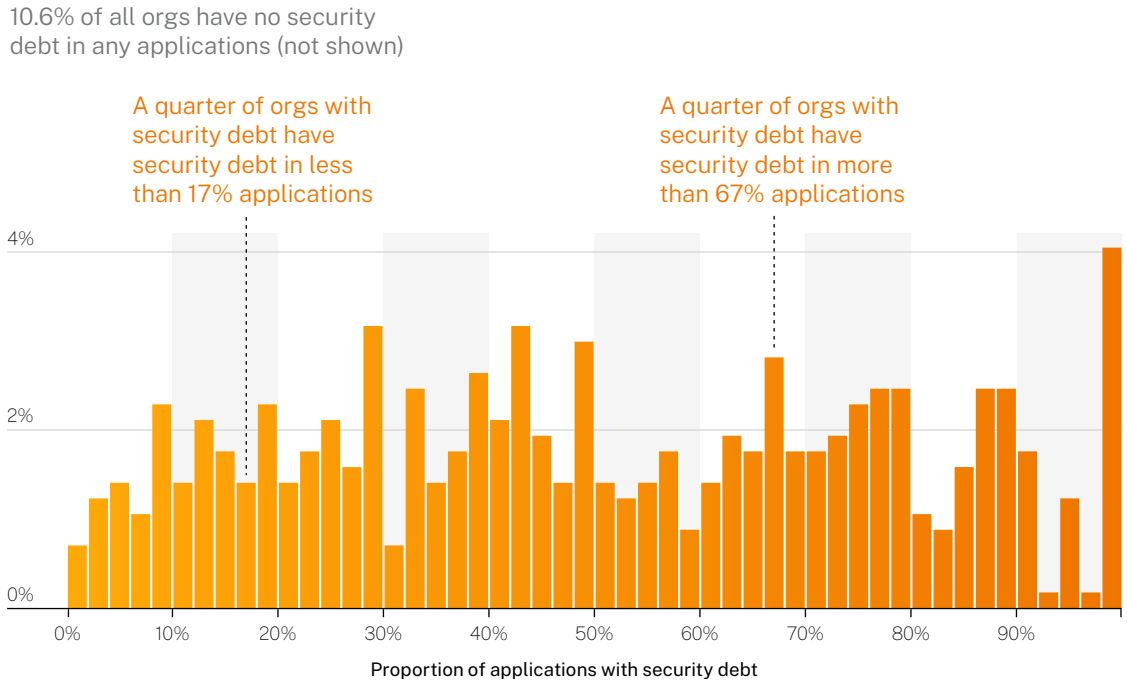
Where do we rank?

Let's start with the good news: Just over 10% of organizations have no security debt. And that's not just because they don't have any flaws to start with—all of them do (see Figure 14). They're clearly doing something right, and we've shared insights gleaned from them along with our security experts in the conclusion.

FIGURE 21

Security debt prevalence among organizations

Percent of organizations



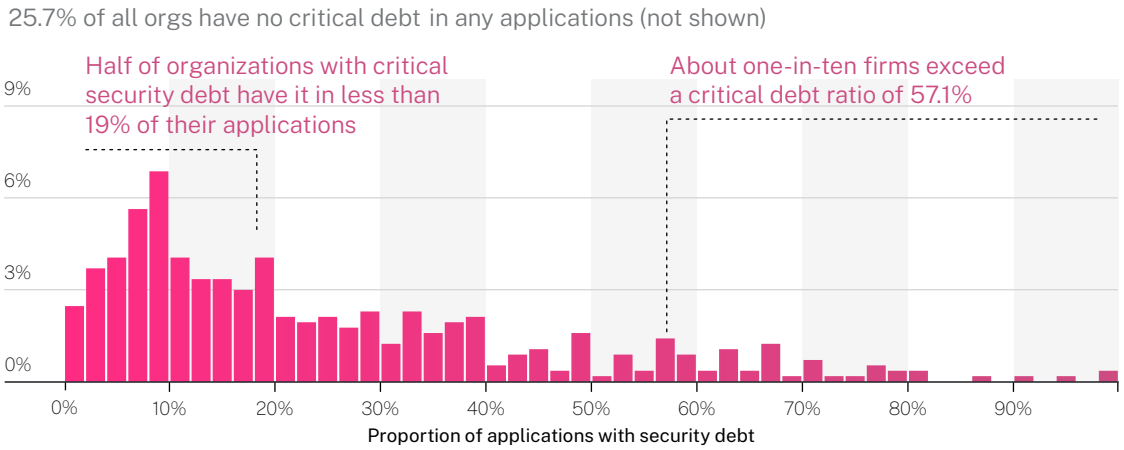
Among organizations that do exhibit some level of security debt, a quarter restricts it to less than 17% of their applications. If you can't be completely debt-free, that's a good goal to shoot for. Lagging organizations struggle with a debt plaguing two-thirds of their applications or more.

Since critical security debt focuses on the riskiest and most persistent flaws, some may wish to track it as a standalone KPI. Figure 22 shows that about a quarter of organizations have no critical debt at all, while a smaller minority exhibit long-unresolved, high-severity issues in over half of their applications.

FIGURE 22

Critical security debt prevalence among organizations

Percent of organizations



What else should I know?

If your organization is not one of the fortunate few to be free of all security debt, it will help to know where it tends to hide so that you can find and eliminate it. Figure 23 breaks down security debt based on the age and size of applications. While debt exists across all categories, it's most concentrated in older, larger applications. This finding hints at a relationship between software security debt and broader forms of tech debt that degrade productivity, efficiency, and resilience in so many organizations.

FIGURE 23

Distribution of security debt across application age and size

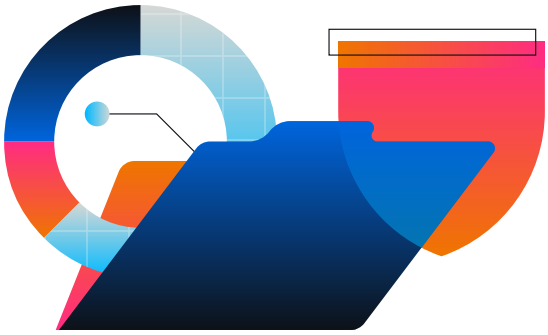
ALL SECURITY DEBT

	YOUNGER (30.1%)	MIDDLE-AGED (33.1%)	OLDER (36.8%)
LARGER (40.4%)	10.6%	12.2%	17.6%
MEDIUM (35.8%)	10.9%	12.6%	12.3%
SMALLER (23.9%)	8.7%	8.3%	6.9%

CRITICAL SECURITY DEBT

	YOUNGER (29.0%)	MIDDLE-AGED (31.7%)	OLDER (39.3%)
LARGER (46.6%)	11.5%	13.1%	22.1%
MEDIUM (30.9%)	9.6%	10.9%	10.4%
SMALLER (22.5%)	7.9%	7.7%	6.8%

NOTE: The charts above show the distribution of security debt among applications grouped into similar age and size ranges. On the age scale, applications are considered younger if they're between 1 and 2.1 years old, middle between 2.1 and 3.4 years, and older after 3.4 years. Those are admittedly odd breakpoints, but they roughly divide all applications into three equal bins. We took a similar approach for grouping small (<250kB), medium (250kB-1.55MB), and large (1.55MB+) applications based on the size of their codebase.



Open-source debt

What is it?

Open-source debt measures the percentage of all security debt that exists in third-party libraries and other software developed outside the organization.

Why does it matter?

We think this is worth differentiating from security debt because what's required to address flaws in third-party code is very different from software written by internal teams. For example, many open-source libraries are dependent on a single contributor who isn't motivated to update their code in a timeframe that's consistent with your risk tolerance and needs.

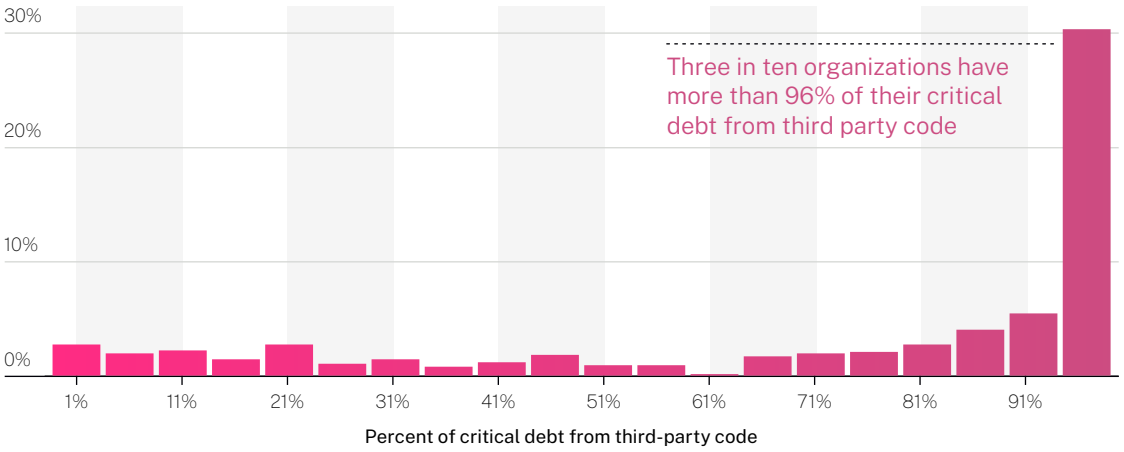
Where do we rank?

The proportion of security debt tied to open-source code is actually fairly low. When it comes to critical security debt, however, a very different picture emerges in Figure 24. The majority of an organization's critical security debt exists in third-party code. Teams on the low end keep that proportion under 15%, while over a quarter of organizations live in the strange reality where ALL of their critical debt is contained in open-source libraries!

FIGURE 24

Prevalence of critical open-source debt among organizations

Percent of organizations



What else should I know?

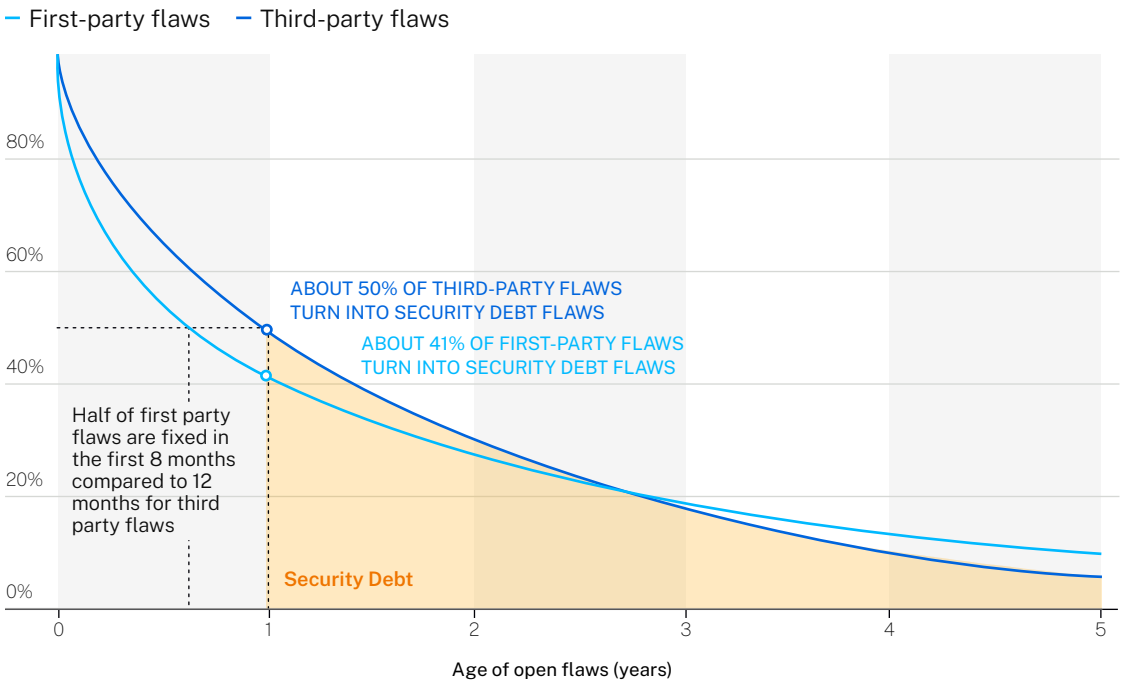
It generally takes longer for flaws to be fixed in open-source code. The survival analysis depicted in Figure 25 reveals that the half-life of flaws in third-party code is 12 months, compared to 8 months in first-party software. This supports the earlier premise that mitigating debt in open-source libraries requires a different strategy.

Curiously, there's a shift in momentum before the three-year mark. We suspect that first-party flaws that haven't been addressed by this point have been risk-accepted or otherwise back burnered, allowing the eventual updates of open-source libraries to catch up.

FIGURE 25

Comparison of remediation timelines for first-party vs. third-party code

Probability flaw is still open



NOTE: Open source flaws come in two types of dependencies: direct and transitive. When your configuration file references a library, it's considered a direct dependency. If those direct dependencies depend on other libraries, they're transitive. Direct dependencies are the easiest to fix. Things get trickier with transitive dependencies; it may be that a fix will break some functionality in the direct library, meaning a slower and more difficult fix process. In some cases there will be code refactoring which takes more time.

Conclusions & Recommendations

The new view of software security maturity is a two-fold perspective that will have a significant impact on your backlog. To mature your software security program efforts in a way that aligns with business objectives, you need:

1. Visibility and integration across your SDLC to prevent net new flaws through automation and feedback loops

The first part of the approach is about having visibility into your SDLC to minimize flaws at the source. You gain visibility into the security of new applications through the automation of continual scanning as developers write their code. This gives you visibility into what's being introduced into applications before they go into production. This is the most cost-effective time to remediate flaws, and remediation is a great use case for responsible-by-design AI. Dealing with flaws as they come in is one of the primary signs of a mature AppSec program.

Artificial Intelligence

Building a sustainable process for continual remediation is much more achievable thanks to recent developments in AI. A large proportion of security debt stems from relatively simple flaws that AI can effectively address at scale. The best teams use these capabilities to their advantage to boost fix capacity and speed.

Policy

Policy is critical for the automation of remediation in the SDLC, because it directs what needs to be fixed. The percentage of apps passing the OWASP Top 10 increasing 63% in 5 years indicates that policy works to drive down risk in the SDLC, as many programs use OWASP as a guidance when setting their policy.

Malicious Package Detection

Third-party flaws are also a huge contributor to the buildup of flaws that pile up into security debt. Evaluating open-source libraries and avoiding those riddled with flaws before importing them into your codebase can slash major issues across applications. A package manager firewall for analyzing, detecting, and mitigating malicious packages before they leave the build is another way to use policy to prevent risky components from ever being brought into an organization. All of this works together to prevent net new flaws, but what about the ones that already exist in the debt that has piled up? We know that the number of organizations with security debt has risen from 71% to 74%. Plus, attack surface complexity has increased, and a third-party library that's secure today may not be secure tomorrow. What can be done to manage this risk? That's why we've expanded the view of software security maturity to incorporate a second part to the perspective: a single view of correlated findings.

2. The ability to correlate and contextualize findings in a single view so you can burn down the backlog based on context and reduce the most risk with the least effort.

Since the average number of days to fix flaws has increased 47% in 5 years, a program that wants to improve security posture and align with business objectives focuses on the findings that matter in context. This is easier said than done due to the ever-growing scope and complexity of the software ecosystem.

As the saying goes, if everything is a priority, nothing is a priority. Your AppSec tools are flooding you with information about what's severe, but you need a way to see what's exploitable, reachable, and urgent to help you prioritize further. To do this, you need visibility from an open and tool-agnostic Application Security Posture Management solution.

Once you have this prioritization, we recommend allocating a percentage of a security champion's sprint capacity to your prioritized security debt and training them on how to make the fix or use AI (by including training time in sprint points, too).

Modern software security is about remediating real risk which requires contextualizing more. We've seen a lot of changes in 15 years of special SoSS, but the rapid proliferation of the attack surface is one that has required us to add to our view of maturity, and we hope you'll do the same.

Methodology

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. Specifically, the data comes from:

- 1.3M unique applications with 126.4M raw findings
- 107.4M findings identified via SAST scans
- 3.9M findings identified via DAST scans
- 15M findings identified via Software Composition Analysis

This data represents companies of all sizes, commercial software suppliers, software outsourcers, and open-source projects.[5] In most analyses, an application was counted only once, even if it was submitted multiple times as vulnerabilities were remediated and new versions were 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.

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 fewer than 2 findings), 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 0.01% accounted for over 1 out of 10 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.

5. 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.

VERACODE