STATE OF SOFTWARE SECURITY

2016

DEVELOPER VIEW ON APPLICATION SECURITY

VERACODE

EXECUTIVE SUMMARY

KEY TAKEAWAYS

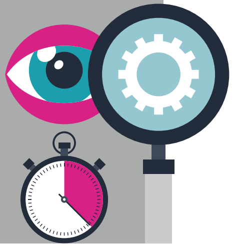
1 Secure code rarely happens on its own.

In their raw state, without any testing or remediation work, about six in 10 applications fail to meet minimum industry standards for security hygiene.

✗ ✗ ✗ ✗ ✗ ✗ ✓ ✓ ✓ ✓

2 Information leakage is a bigger problem than developers think.

While error messages and other clues that sneak through into userland may not seem like that big of a deal, they're giving attackers valuable information to build better attacks. These vulnerabilities are the number-one type found in applications today.



3 Cryptographic and credentials management problems plague code.

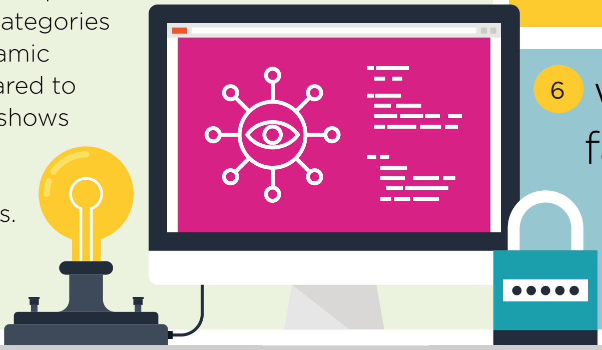
Crypto issues follow closely on the heels as the second most common type of vulnerability, and credentials problems are in the top 10.

About **35%** of applications use hard-coded passwords

and **39%** use broken or risky crypto algorithms.

4 There's no silver bullet for security testing. **Catching a range of flaws requires different kinds of testing.**

Comparing the top 15 vulnerability categories found by dynamic testing compared to static testing shows some overlap, but there are big differences.

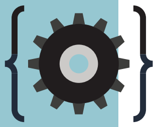


5 Developers aren't getting security features right.

Looking at the top 15 vulnerability categories found by dynamic testing, four of them are the result of improper application of security mechanisms.

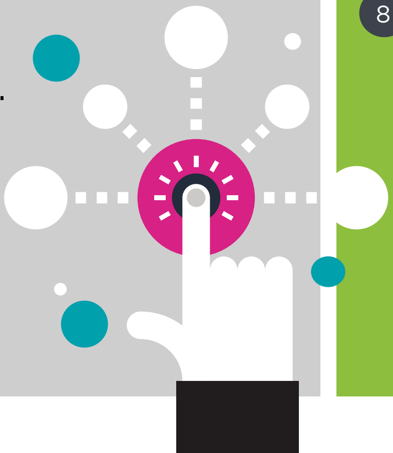
6 Web scripting languages fail more frequently.

Web scripting languages tend to have a higher fail rate upon first scans, as compared to more mature enterprise application languages like Java or .NET



7 More frequent testing results in better AppSec.

When developers are given a way to test applications more frequently without compliance minders nitpicking results, the average fix rate tends to double as a result. Other best practices like remediation coaching and eLearning can improve vulnerability fix rates by as much as 6x.



8 DevOps is bolstering continuous testing.

Fortunately, the DevOps movement's emphasis on frequent iteration seems to be pumping up the level of testing frequency at a number of organizations. The top range witnessed was as many as six security scans per day.

OVERVIEW

As the ties between application developers and security professionals grow closer than ever through the growing shift to DevOps and continuous testing, **developers are increasingly tasked with more hands-on security testing and mitigation work.** While security teams remain a steadfast ally and consultant in application security success, the fact is that the typical developer will need to increase their AppSec IQ to remain relevant and competitive in the changing application development environment.

Now in its seventh volume, the Veracode State of Software Security (SOSS) report has long stood as a valuable resource for security professionals looking for benchmarks and analysis to measure their organizations' performance against. The data contained within these reports offer a lot of valuable information for developers seeking to improve their security game. But geared for a different audience as they are, they can be a lot to wade through for a busy developer seeking out only the most relevant information for their purposes.

The purpose of this supplement to the State of Software Security volume 7 is to boil down the data analysis to the key takeaways developers will be most interested to see.

The purpose of this supplement to SOSS v7 is to help save developers time, boiling down the data and the analysis to the points developers will be most interested to see. Everything is statistically backed in the main report; the data has just been abridged and reframed to give developers the perspective they need to improve their practices and avenues for pursuing further training in the future.

The first two sections take a look at the statistics and insights that bubbled up from the main body of the full report. For those looking for more context or to answer data-related questions, we encourage you to read the full SOSS report at length.

The third section offers some supplemental data not found in the main report, giving developers a more in-depth look at vulnerability management trends as framed by programming languages. These statistics should help developer teams anticipate the types of vulnerabilities that are likely to arise when selecting languages and methodologies for new projects in the future.

We conclude with some statistically backed best practices that can help developers improve their AppSec game. These are the practices that we've seen can offer the most measurable improvement for the effort involved.

VULNERABILITY BENCHMARKS

As we explore more detailed data and trends in this supplemental report, it'll help to frame them with some of the most important high-level benchmarks to come out of the main SOSS analysis this year, paired with some relevant data from a few other recent industry studies for greater context.

The conclusions we draw here offer a good jumping off point for developers serious about finding meaningful yardsticks against which they can measure their own security performance.

PASS RATES

The raw state of untested code tends to remain about the same year after year. When examining software for the first time for security, typically a little less than two-thirds of applications fail to meet minimum industry standards for secure code. That holds true this year: Fewer than four in 10 applications passed standards test for either the OWASP Top 10 or CWE/SANS Top 25.

Again, that's software that has never gone through any kind of security review. It includes legacy code that never went through security testing in the past, along with new code that didn't undergo continuous testing as part of a well-rounded secure development lifecycle (SDLC).

While there are no good industry statistics available to directly show what percentage of the global software code base remains in this raw state, some smart inferences can show that the number is still exceedingly high.

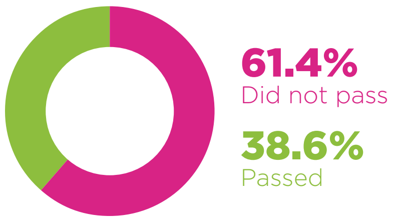
According to a SANS Institute study this year, just shy of a quarter of organizations would characterize their application security programs as mature or very mature.

Based on our work with organizations across the maturity spectrum, Veracode can confirm that even the cream of the crop has only tested a minority percentage of its code base for security.

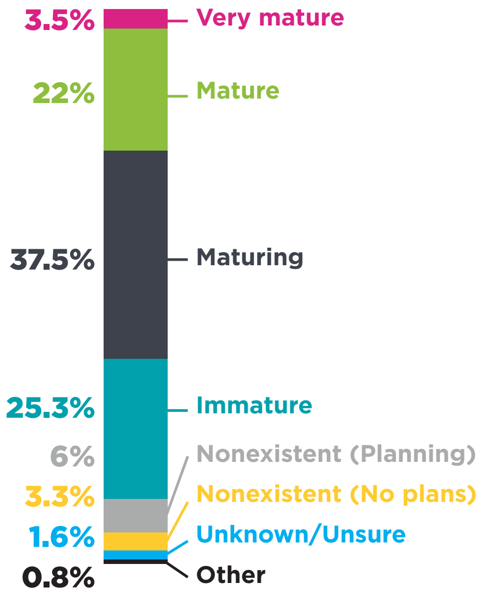
This means that **most code today still stands in this raw state of insecurity.**

The takeaway here for developers is that secure code infrequently happens on its own. It takes constant work to achieve. That includes keeping secure coding best practices top-of-mind during development, testing throughout the SDLC and continually refining code based on the results of those tests and the risk appetite of the business.

Percentage of applications passing OWASP Top 10 policy



How mature do you consider your AppSec program to be?

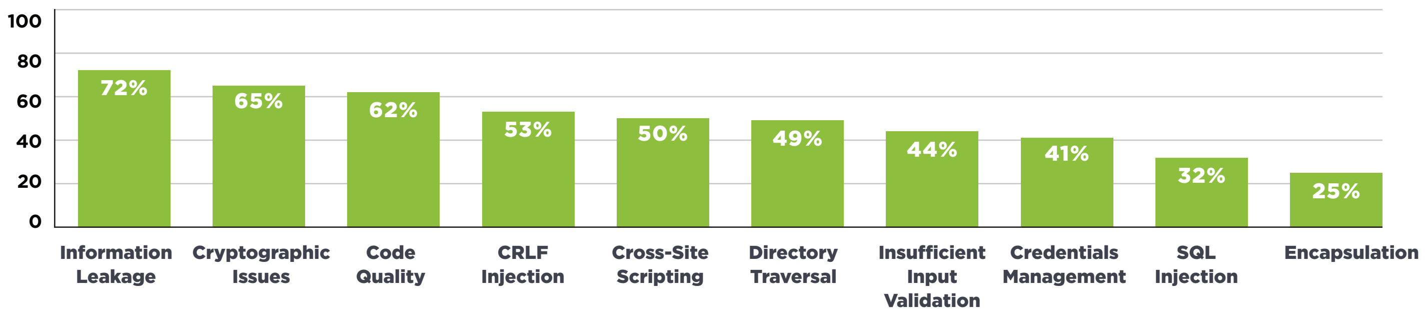


VULNERABILITY PREVALENCE BY TYPE

Our SOSS data annually provides some important observations that can help developers understand the kinds of coding mistakes most likely to compromise the security of their software.

As in years past, we broke down vulnerabilities based first on the MITRE common weakness enumeration (CWE) taxonomy and then rolled those up into a set of 30 broad categories. From there, we ranked the top 10 vulnerability categories to crop up based on a combination of both static and dynamic tests across over 330,000 assessments during an 18-month period.

Top 10 vulnerability categories overall



There are some useful observations from this list that can offer developers immediate direction on how to improve their security knowledge and coding habits moving forward:

1 Developers are getting better about SQL Injections (SQLi) and Cross-Site Scripting (XSS). SQLi and XSS still remain a thorn in the side of a lot of software—just not to the degree that they did in the past. Even though attacks against these vulnerabilities garner a lot of media attention when they happen, the statistics show that they aren't necessarily the most common vulnerabilities floating around in code today. Nevertheless, they're still critical.

2 It's time to toughen up with information leakage. Letting error messages and the like pop up in userland with key tidbits about the application, the environment or user data may not always seem like a big deal. But cybercrooks thrive on these breadcrumbs and use them to eventually build bigger and better attacks. Information leakage vulnerabilities open software up to a death by a thousand cuts. And based on prevalence that they exist, it's a particularly bloody one at that.

3 Cryptographic and credentials management problems are a dark cloud on a huge percentage of applications today. Those overall percentages reflect apps afflicted with issues like these with varying degrees of severity. But our SOSS analysis also shows that the most serious ones happen quite frequently—35% of all applications use some kind of hardcoded password and 39% use broken or risky cryptographic algorithms. And anecdotally some of the biggest vulnerability and zero-day attack scares of the last 18 months have come from cryptographic problems—both DROWN and HEIST are good recent examples of this.

INSIGHTS FOR DEVELOPERS

One of the big drivers for putting out this supplement was to give developers a chance to "skip to the good parts" of the SOSS without necessarily having to wade through the entire data set or sweat the minutiae of how things were analyzed.

Here's a grab bag of some of the most important insights developers can glean from the juiciest data points presented in the body of the main report.

GETTING TO REMEDIATION

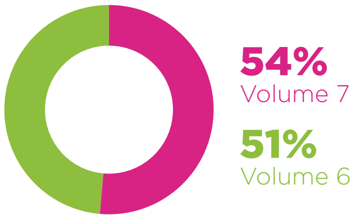
Ultimately, the goal of security testing is to put the right tools in the hands of security and development teams so they can fix flaws as efficiently as possible. This year's SOSS data offered up some decent metrics that gauge how well development teams are remediating vulnerabilities, along with some factors that can improve their effectiveness on that front.

First, as a level-set, we found that the typical fix rate of vulnerabilities discovered in the first scan compared to those revealed in the last scan was about 54%—that's accounting for vulnerabilities that were fixed and later reopened in future tests. This is only the second year of collecting this metric, but we did see a measurable improvement by about three percentage points in the past year.

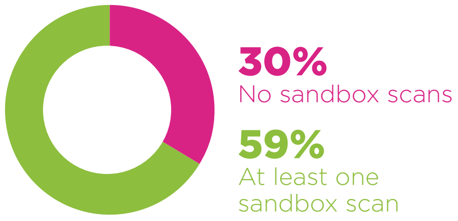
Fix-rate data allows us to illustrate the fact that frequent testing can greatly improve remediation efforts. The data in the image to the right is based on compliance-driven policy scans that are shared with security personnel and other stakeholders. But Veracode also offers the capability for developers to test in a sandbox mode that gives them a chance to perform regular assessments that are effectively "off-the-record." This gives developers the freedom to test and improve code incrementally without anyone breathing down their necks about poor early results.

The measurable effects of this kind of testing are undeniable. Even just a single sandbox scan improved an organization's software fix rate almost two-fold.

Average fix rates



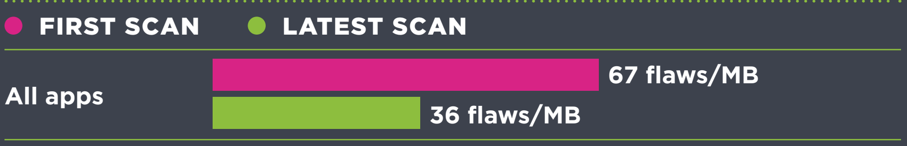
The Fix-Rate Effects of Sandbox Testing



In addition to frequent reassessments, other favorable practices went a long way toward helping developers improve their rate of remediation. We were able to discover these through the use of flaw density metrics. Flaw density provides a way to make an apples-to-apples, before-and-after comparison of an application's incidence of vulnerabilities by keeping track of security bugs per MB of code. This way you can show changes over time while still accounting for the addition or subtraction of functionality, libraries and the like.

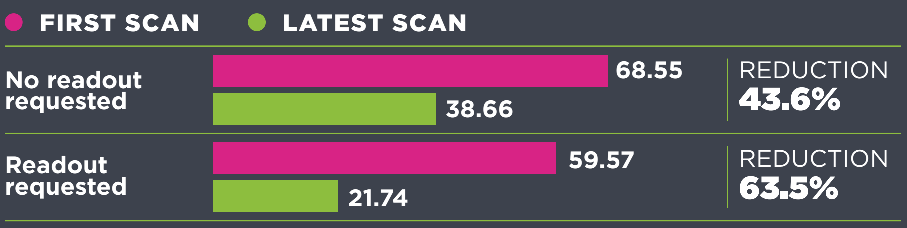
On average, flaw density is typically almost cut in half between the first assessment and the last reassessment at any given organization.

Reduction in flaw density - first assessment vs. reassessment



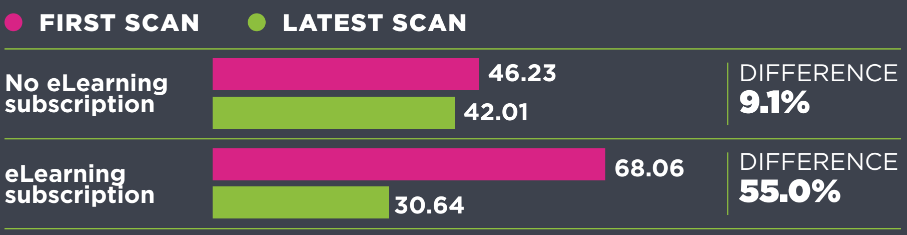
But there are several practices that can greatly improve flaw density metrics during remediation. For example, remediation coaching improves flaw density reduction by about 1.45x.

Reduction in flaw density via remediation coaching



And even more dramatic, eLearning and developer training go even further, giving organizations six times better flaw density reduction metrics.

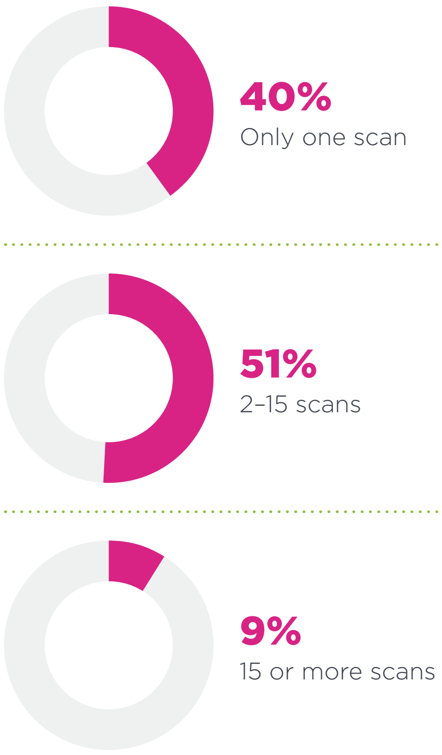
Reduction in flaw density via eLearning

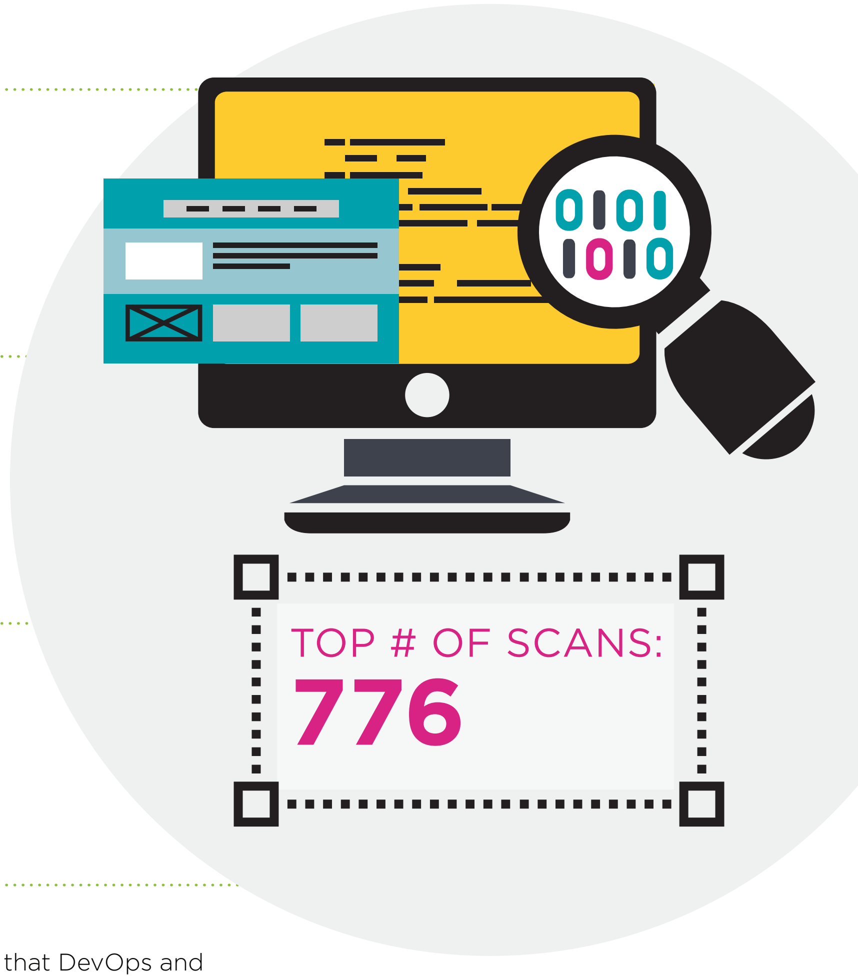


DEVOPS AND CONTINUOUS TESTING

As we emphasized above, the incremental improvements afforded by frequent reassessments can provide a huge boost to organizations' remediation rates. This latest volume of SOSS showed that the majority of applications today just get the bare number of rescans necessary to get them into policy compliance. The average number of scans per app is seven and the median is two.

Rate of policy reassessment in 18-month period





However, there are some really good signs popping up that DevOps and continuous testing practices are starting to creep into a statistically relevant number of organizations. As things stand, about 9% of applications get more than 15 policy scans within an 18-month period. But the telling thing was that on the upper range, there were some organizations doing as many as 776 policy scans during that same time period—about 1.4 official scans per day. And when accounting for scans in sandbox mode, that grew to as many six scans per day.

This kind of frequency suggests strongly that certain organizations are moving to the type of DevOps and continuous delivery patterns that go a long way toward moving the needle on remediation fix rates and flaw density reduction.

A PROOF POINT FOR VARIED TESTING

We hope, too, that the increasing frequency of testing is also balanced with a broadening scope of testing methods. This year's SOSS data offered up a good statistical proof point for why multi-dimensional testing is so important for finding and fixing a broad range of software vulnerabilities.

In addition to examining the top 10 vulnerabilities based on a composite of dynamic and static testing, the team also analyzed the top vulnerabilities found by each kind of testing.

Top 15 vulnerabilities: Dynamic vs static

CWE CATEGORY	DYNAMIC RANK	DYNAMIC % APPS AFFECTED	STATIC RANK	STATIC % APPS AFFECTED
Information Leakage	1	86%	1	69%
Cryptographic Issues	2	58%	3	65%
Deployment Configuration	3	57%	na	na
Encapsulation	4	39%	11	22%
Cross-Site Scripting (XSS)	5	25%	6	52%
Credentials Management	6	15%	8	43%
Session Fixation	7	12%	15	15%
Server Configuration	8	10%	24	3%
SQL Injection	9	6%	9	35%
Authentication Issues	10	4%	23	na
Insufficient Input Validation	11	3%	7	48%
Directory Traversal	12	2%	5	54%
Code Injection	13	1%	21	4%
Command or Argument Injection	14	1%	13	15%
CRLF Injection	15	<1%	4	59%

At a glance, it is obvious that while there's crossover for each type of scan, there are also considerable differences in what gets found when scanning dynamically at runtime versus testing in a non-runtime environment. The point here is that neither test is necessarily better than the other, they're just different. The only way to cover all your bases is to remember that there's no silver bullet for security testing.

SECURITY FEATURES ARE BEING APPLIED INCORRECTLY

Developers could also stand to look at the chart in the above section one more time and focus on the top vulnerability categories found by dynamic testing. There's another takeaway hidden there, and it has to do with security features. Namely that in the quest to bolster security within their applications, many developers are opening up their software to even more attacks.

All four of the top vulnerability categories in the dynamic testing top 15 could be bundled together into a broader grouping of vulnerabilities that stem from improper application of security mechanism. Whether it is utilizing SSL incorrectly, accepting the wrong cross-domain policies or improperly using security headers, these types of flaws increase the attack surface area of software while engendering a false sense of security. It's a wakeup call to developers on the importance of working with security professionals to ensure that they're properly instituting these features within their software.

COMPONENT VULNERABILITY DANGERS

The security and development community is finally starting to get serious about the dangers posed by vulnerable components and libraries lurking in software today. Veracode wanted to contribute a bit of its own data and analysis on the topic, which is why we dedicated quite a bit of room in this volume of SOSS examining common vulnerabilities found in Java components.

The high-level synopsis is that **97% of all Java applications assessed this time around had at least one component with a known vulnerability.** The most prevalent vulnerability was a Java deserialization vulnerability found in Apache Commons Collections that was newly patched midway through the 18-months examined by SOSS analysis. So it wasn't surprising that it was found in 30% of all Java applications.

The troubling thing was that the second most common component vulnerability was actually patched way back in 2010 and was nevertheless found in over 12% of all Java applications. This shows that a startling number of organizations are falling down at the job of keeping their components updated and secure.

APPLICATION DEVELOPMENT LANDSCAPE

Amid the slew of useful information offered up in the main SOSS report, we left out a few key pieces of data that we felt might seem superfluous to the security wonks in the crowd. But we knew they'd be of keen interest to developers. Putting this additional information in front of the development community was also part of the impetus behind this supplement.

These data points provide a valuable glimpse into today's preferences for programming languages in the enterprise, along with breakdowns on how prone certain languages are to security imperfections.

PROGRAMMING LANGUAGE PREVALENCE

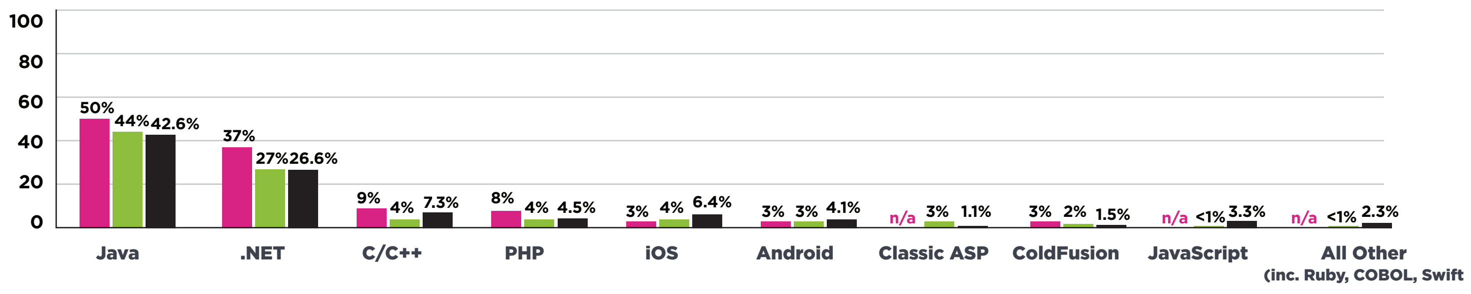
Let's start first by looking at how programming language choices have shifted in the last several years. Many of the changes are subtle, but they do provide some visibility into overall changes occurring in the enterprise development landscape.

Over the past three years, the relative ranks for these languages have remained largely unchanged, but there are some new entrants into the field and they're becoming statistically significant fairly quickly. The two juggernauts, Java and .NET, remain at the top of the heap. Though their percentages have dipped slightly over the last three years, we believe it was less a matter of them losing share and more the fact that Veracode added coverage to JavaScript, which naturally shifted the pieces of the pie.

Speaking of JavaScript, the prevalence of this language is growing faster than any other in Veracode's recent history. Our measurements only include Node.JS and cross-platform mobile applications, but in just a couple of years, it has grown to make up over 3% of the assessed code base. Meanwhile, iOS and Android continue to grow in share at a steady clip as well.

As for legacy scripting language, while PHP did make a very slight gain, it stays deflated compared to its peak popularity in our SOSS Volume 5 data set. And both Classic ASP and Cold Fusion are down since the last data set was analyzed.

Programming language distribution ● **VOLUME 5** ● **VOLUME 6** ● **VOLUME 7**



For developers who want a sense of how things look for languages on an industry-by-industry basis, we've also broken this down by major verticals.

There are a couple of insights we can draw from the differences in language composition between different industry segments. First of all, there appears to be a much more rapid adoption of JavaScript for server-side and mobile use in healthcare, retail and financial services—this could perhaps point to a move to modernize software in industries known to lean heavily on legacy applications.

Secondly, there also appears to be a larger investment in mobile technologies in both the technology and retail sectors. Tech is an obvious mobile beachhead, and many retailers are likely jumping on the mobile bandwagon due to rising consumer expectations and open opportunity to increase revenue shares.

There appears to be a more rapid adoption of JavaScript for server-side and mobile use in healthcare, retail and financial services.

Programming language breakout by industry vertical

PROGRAMMING LANGUAGE	FINANCIAL SERVICES	MANUFACTURING	TECH	RETAIL AND HOSPITALITY	OTHER	GOVERNMENT	HEALTHCARE
.Net	38.33%	33.76%	27.59%	40.68%	27.66%	52.20%	46.90%
Java	47.29%	39.09%	46.86%	37.24%	42.10%	35.05%	25.22%
C++	1.46%	8.39%	3.40%	2.70%	3.18%	0.00%	0.00%
ColdFusion	0.14%	7.28%	0.45%	0.34%	0.73%	1.52%	0.88%
Javascript	4.98%	2.62%	5.74%	7.77%	6.92%	3.49%	11.50%
Android	1.38%	1.09%	5.48%	3.38%	4.25%	0.46%	3.54%
iOS (Objective-C)	1.52%	1.71%	4.82%	3.38%	4.60%	0.00%	7.52%
Classic ASP	1.33%	3.42%	0.45%	0.96%	0.30%	1.97%	0.44%
PHP	2.12%	1.42%	4.34%	2.37%	9.49%	5.16%	1.33%
VB6	0.84%	0.91%	0.22%	0.34%	0.43%	0.15%	0.00%
COBOL	0.48%	0.05%	0.18%	0.39%	0.00%	0.00%	0.00%
Ruby	0.03%	0.08%	0.16%	0.06%	0.00%	0.00%	0.88%
Other (J2ME, Windows Mobile)	0.09%	0.16%	0.29%	0.39%	0.35%	0.00%	1.76%

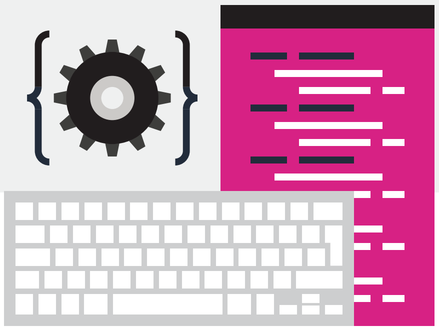
VULNERABILITY BENCHMARKS BY PROGRAMMING LANGUAGE

Now that we've covered overall vulnerability benchmarks and the popularity of different programming languages in the enterprise, let's look at how vulnerability benchmarks break out by programming language.

SPECULATION ON C/C++ HIGH RATE OF PASSING?

Policy compliance by programming language

✓ % PASS OWASP		✗ % FAIL OWASP
61.3%	Java	38.7%
61.9%	.NET	38.1%
87.9%	C/C++	12.1%
32.5%	PHP	67.5%
64.4%	iOS	35.6%
64.2%	Android	35.8%
41.9%	Classic ASP	58.1%
37.5%	ColdFusion	62.5%
40.5%	JavaScript	59.5%



Pass Rates

The data here is fairly straightforward. As you can see in the information to the left, software programmed in web scripting languages tend to struggle more to meet OWASP standards right out of the gate. This likely has to do with the fact that languages like Java and .NET are enterprise-grade languages, with a more mature ecosystem and more stable base of programmers trained with a security mindset from the get-go.

Interestingly, both iOS and Android have significantly heightened rates of compliance with OWASP standards. This may have less to do with the exemplary nature of those code bases and more to do with the still-maturing standards for mobile development environments.

Vulnerability Prevalence

Yet again, we've gathered a significant body of data around the types of vulnerabilities most likely to crop up within the most popular programming languages. The information In the table on page 14 offers a nice reference for developers specializing in one of these languages who'd like to measure their performance with a fair comparison against similar software.

Each language has its own quirks in vulnerability prevalence, but the overarching theme we noticed here is that web scripting languages like Classic ASP, ColdFusion, PHP and even Ruby tend to suffer from higher-than-average incidence of XSS and SQLi. The exception is JavaScript.

However, we'd warn that this may have to do with the way we collected data in this volume of SOSS, where we've lumped together both web server and mobile applications in one group. We plan to remedy this situation in future volumes, so keep an eye out next time around for how that affects the numbers.

Top 10 vulnerability categories by programming language

LANGUAGE/CWE CATEGORY		% APPS
.Net	Information Leakage	73%
	Code Quality	68%
	Cryptographic Issues	67%
	Directory Traversal	61%
	Insufficient Input Validation	57%
	Cross-Site Scripting (XSS)	51%
	CRLF Injection	44%
	SQL Injection	37%
	Credentials Management	28%
	Untrusted Initialization	22%
Android	Code Quality	93%
	Cryptographic Issues	91%
	CRLF Injection	87%
	Information Leakage	84%
	Insufficient Input Validation	52%
	Directory Traversal	47%
	Credentials Management	40%
	SQL Injection	39%
	Time and State	32%
	Encapsulation	31%
ASP	Cross-Site Scripting (XSS)	85%
	SQL Injection	68%
	Information Leakage	63%
	Insufficient Input Validation	57%
	Directory Traversal	46%
	Credentials Management	34%
	CRLF Injection	30%
	Cryptographic Issues	27%
	Session Fixation	25%
	Command or Argument Injection	22%
C++	Error Handling	79%
	Buffer Management Errors	64%
	Buffer Overflow	63%
	Numeric Errors	58%
	Cryptographic Issues	58%
	Directory Traversal	53%
	Potential Backdoor	42%
	Code Quality	40%
	Race Conditions	35%
	API Abuse	32%
COBOL	Error Handling	57%
	Other	51%
	Code Quality	31%
	Information Leakage	23%
	Credentials Management	18%
	Directory Traversal	12%
	Cryptographic Issues	8%
	Authorization Issues	8%
	CRLF Injection	8%
	SQL Injection	8%
ColdFusion	Cross-Site Scripting (XSS)	84%
	SQL Injection	59%
	Information Leakage	58%
	Code Quality	36%
	Directory Traversal	36%
	Time and State	33%
	CRLF Injection	3%
	Cryptographic Issues	2%
	Command or Argument Injection	1%
	Session Fixation	1%

LANGUAGE/CWE CATEGORY		% APPS
iOS	Error Handling	86%
	Cryptographic Issues	82%
	Credentials Management	69%
	Potential Backdoor	62%
	Information Leakage	61%
	API Abuse	17%
	Code Quality	16%
	Coding Standards	10%
	Buffer Overflow	9%
	Directory Traversal	6%
Java	Code Quality	84%
	CRLF Injection	83%
	Information Leakage	71%
	Cryptographic Issues	65%
	Cross-Site Scripting (XSS)	58%
	Credentials Management	55%
	Directory Traversal	55%
	Insufficient Input Validation	52%
	Encapsulation	44%
	Time and State	38%
JAVASCRIPT	Cross-Site Scripting (XSS)	35%
	Information Leakage	27%
	Insufficient Input Validation	23%
	CRLF Injection	21%
	Credentials Management	20%
	Cryptographic Issues	20%
	Directory Traversal	17%
	Code Quality	12%
	Code Injection	10%
	SQL Injection	10%
PHP	Cross-Site Scripting (XSS)	87%
	Cryptographic Issues	80%
	Directory Traversal	75%
	Information Leakage	67%
	Untrusted Initialization	67%
	Credentials Management	66%
	Code Injection	62%
	SQL Injection	53%
	Command or Argument Injection	45%
	Insufficient Input Validation	16%
Ruby	CRLF Injection	60%
	SQL Injection	55%
	Insufficient Input Validation	55%
	Cross-Site Scripting (XSS)	45%
	Directory Traversal	35%
	Information Leakage	30%
	Cryptographic Issues	20%
	Untrusted Initialization	15%
	Code Injection	10%
	API Abuse	5%
VB6	Information Leakage	72%
	Credentials Management	68%
	Error Handling	55%
	Cryptographic Issues	45%
	SQL Injection	40%
	Directory Traversal	39%
	CRLF Injection	21%
	Command or Argument Injection	14%
	Cross-Site Scripting (XSS)	10%
	Authorization Issues	9%

To really get to the heart of the most severe problems for each language, we also zoomed in specifically on the incidence of the most critical vulnerability types. In examining the data here, beyond the observation above about the prevalence of SQLi and XSS by language, we also found that mobile languages and PHP seem to suffer from some of the most severe cryptography and credential management issues.

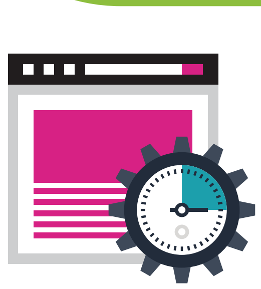
Comparison of critical vulnerability types by programming language

PRIMARY LANGUAGE	CROSS-SITE SCRIPTING	SQL INJECTION	CRYPTOGRAPHIC ISSUES	CREDENTIALS MANAGEMENT
.NET	51%	37%	67%	28%
Android	2%	39%	91%	40%
Classic ASP	85%	68%	27%	34%
C/C++	8%	9%	58%	18%
COBOL	3%	8%	8%	18%
ColdFusion	84%	59%	2%	1%
iOS (Objective C)	0%	1%	82%	69%
Java	58%	35%	65%	55%
JavaScript (NodeJS, Mobile)	35%	10%	20%	20%
PHP	87%	53%	80%	66%
Ruby on Rails	45%	55%	20%	0%
VB6	10%	40%	45%	68%

It's not shown in the chart above, but on the cryptography front, both Android and PHP suffer most from using broken or risky cryptography algorithms. This problem crops up in 37% of Android apps and 49% of PHP apps. Meanwhile, iOS' most serious crypto problem is cleartext storage of sensitive info, which happens in 35% of these apps.

On the credentials management side of things, iOS and PHP are the worst offenders. In iOS, a whopping 88% of these apps suffer from poor use of the Apple Keychain feature. And in PHP, an incredible 95% of software contains some sort of hardcoded passwords.

In PHP, an incredible 95% of software contains some sort of hardcoded passwords.



BEST PRACTICES FOR APP DEVELOPERS

As application security increasingly becomes non-negotiable for upper level managers, more and more developers are going to be measured and judged not just by features and timetables but also by how securely they code their software. The observations provided in this report can provide some excellent fodder to help developers at every level look for ways to up their AppSec game. Here's the payoff for all of the analysis—if there are five recommendations developers should take from this report, it would be the following:

Test Early and Often

Frequent reassessments and incremental improvements help developers fix code faster and more efficiently.

Train Up

Training services tied to principles around commonly discovered vulnerabilities can greatly aid developers to not only fix the found problems, but also avoid making similar mistakes in future programming work.

Don't Be a One-Trick Pony

There is no single test that will find all the risks in an application. A well-rounded approach with multiple test types is required to cast the widest net possible for all types of vulnerabilities.

Keep an Eye on Components

Components are easy to forget about, but vulnerabilities within them are just as dangerous as the flaws within proprietary code. Developers need to be more vigilant about keeping tabs on components and updating them regularly.

Ask for Help

Developers may be shouldering more security responsibilities these days, but that doesn't mean they need to go it alone. Remediation coaching can go a long way toward improving how efficiently developers fix flaws after initial tests. What's more, security teams can be a great resource in helping to properly embed security features within software.

STATE OF
SOFTWARE SECURITY

2016

DEVELOPER VIEW
ON APPLICATION SECURITY

WHITEPAPER

Five Principles for Securing DevOps

DevOps, a new organizational and cultural way of organizing development and IT operations work, is spreading rapidly – driven by mounting evidence of its benefits to the business. But reaping these gains requires rethinking application security to deliver more secure code at DevOps speed.

Are you ready to rethink application security so you can deliver more secure code at DevOps speed? Find out how in our guide, **Five Principals for Securing DevOps**.

VERACODE

SECURING THE SOFTWARE THAT POWERS YOUR WORLD.

Veracode's cloud-based service and systematic approach deliver a simpler and more scalable solution for reducing global application-layer risk across web, mobile and third-party applications. Recognized as a Gartner Magic Quadrant Leader since 2010, Veracode secures hundreds of the world's largest global enterprises, including 3 of the top 4 banks in the Fortune 100 and 20+ of Forbes' 100 Most Valuable Brands.

LEARN MORE AT WWW.VERACODE.COM, ON THE VERACODE BLOG, AND ON TWITTER.