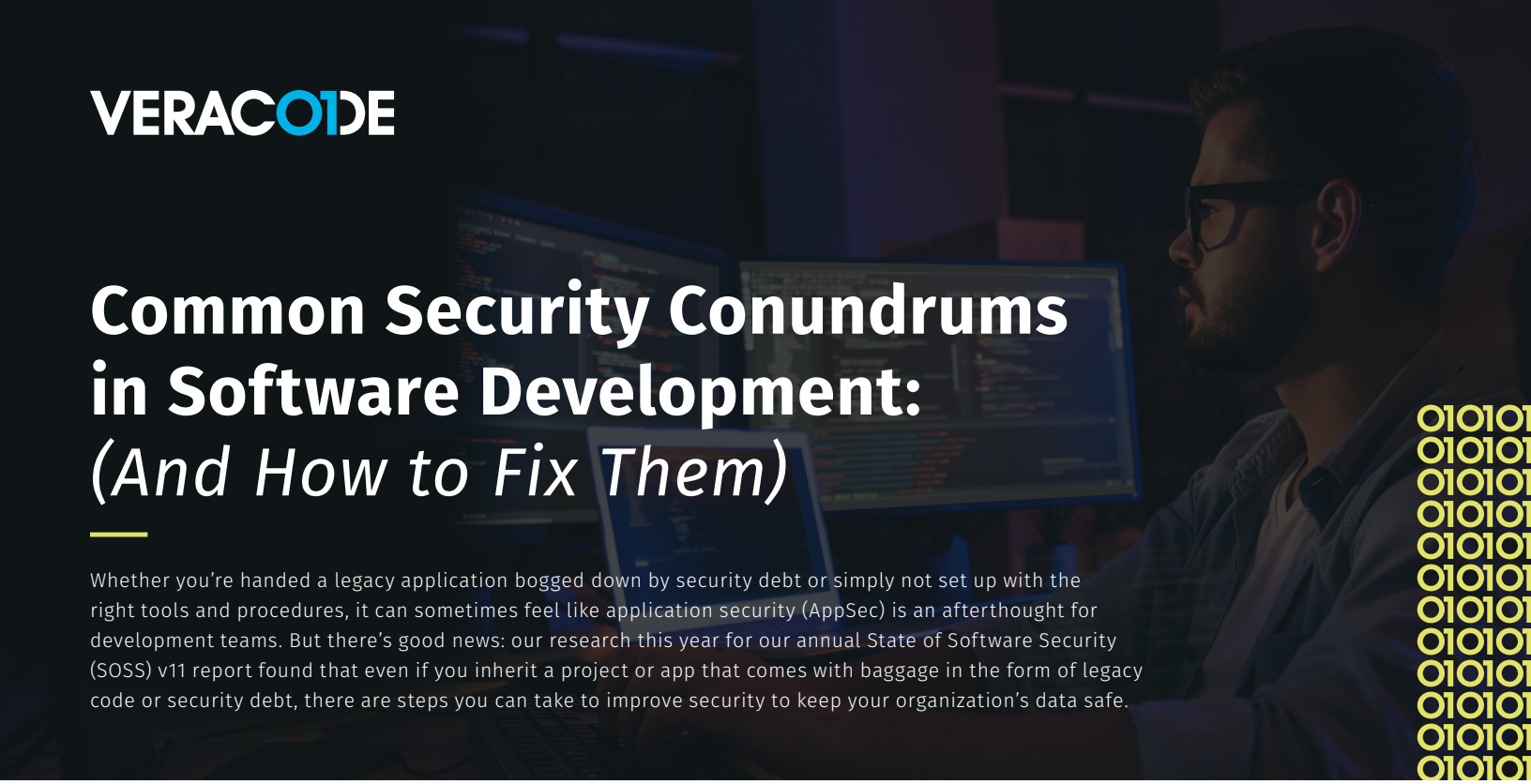
VERACODE



Common Security Conundrums in Software Development: *(And How to Fix Them)*

Whether you're handed a legacy application bogged down by security debt or simply not set up with the right tools and procedures, it can sometimes feel like application security (AppSec) is an afterthought for development teams. But there's good news: our research this year for our annual State of Software Security (SOSS) v11 report found that even if you inherit a project or app that comes with baggage in the form of legacy code or security debt, there are steps you can take to improve security to keep your organization's data safe.

The Problem: Our remediation time for vulnerabilities is greater than 50 days, which slows down our processes.

The Fix: Lingering vulnerabilities are an issue that create and inflate security debt and, in turn, risk. To avoid adding to security debt consider automated scanning and adding testing types. Our research this year found that organizations which pair Dynamic Analysis ***(DAST)*** with Static Analysis ***(SAST)*** fix 50 percent of their security flaws 24.5 days faster on average. One theory for why is that getting a dynamic analysis result switches the vulnerability from theoretical to real-world and incites developers to remediate it faster.

Our research also found that scanning more frequently enables organizations to reach that halfway point 22.5 days faster and running SAST scans through API decreases the time to remediate 50 percent of flaws by 17.5 days. These efforts can vastly improve your median time to remediation (MedianTTR) and help you reduce security debt.

To chip away at existing debt, it's vital that you stay on top of new flaws introduced during development while also prioritizing and remediating lingering issues. By implementing a steady scanning cadence, it's possible to see meaningful change in the proportion of flaw types and reduce security debt over time. This becomes easier as you use scan results in more mature ways, selecting issues to tackle first and weighing the debt you're willing to tolerate in the meantime. Ultimately, that puts you one step ahead of your security debt instead of drowning underneath it.

The Problem: We keep seeing information leakage, CRLF injection, cryptographic issues, and code quality issues.

The Fix: Information leakage (65.9 percent), CRLF injection (65.4 percent), cryptographic issues (63.7 percent), and code quality (60.4 percent) are the most common flaws found in applications, and those four have been in the top 10 for years in our annual State of Software Security reports. Understanding the flaws that pose the greatest risk to your applications – including how they're introduced and how to fix them quickly – is key to preventing expensive and damaging cyberattacks that these common flaws enable. And the fact that these vulnerabilities keep emerging in code year after year highlights an awareness and training gap among developers.

Fortunately, these vulnerabilities are avoidable: to mitigate information leakage, lean on secure coding best practices and implement security testing procedures as you write code. You can prevent CRLF injection by never trusting user input, sanitizing user-supplied data with proper validation and encoding, and by correctly encoding output in HTTP headers. Cryptographic vulnerabilities are preventable with good secure coding practices, but also, most major languages inherently support good cryptographic practices and concerns over incorrect implementation typically arises on a case-by-case basis. Finally, you can prevent poor code quality issues by utilizing consistent coding patterns, automating security testing in your SDLC, and keeping up to date through effective training.

Effective developer training isn't just about checking boxes to meet the bare minimum for compliance; it needs to go beyond show-and-tell to provide actionable, real-world guidance ***that sticks***. When developers learn about preventing real flaws and vulnerabilities through muscle memory, they're ready and able to pivot at a moment's notice while writing code, fixing errors before they become costly coding mistakes.

The Problem: We rely on open source libraries a lot, but we only scan application code that we write in-house.

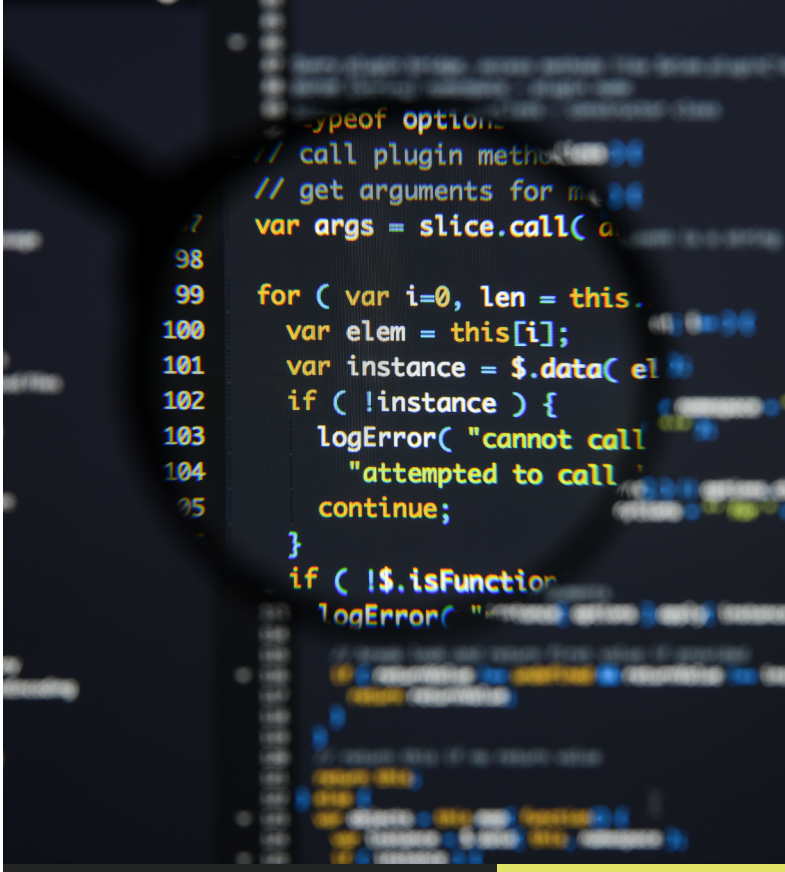
The Fix: It's used virtually everywhere by developers, and so open source code naturally creates an expanding attack surface within applications. This is especially true for specific languages; Java applications are 97 percent third-party code and carry greater unseen risk. We know from ***State of Software Security: Open Source Edition*** that most (71 percent) applications have a flaw in an open source library on initial scan. But there's good news. We also know that most of the required security fixes for these libraries are small; about 74 percent of them can be fixed with a patch, revision, or major/minor version update.

Integrating a scanning tool like Software Composition Analysis (SCA) will help you detect open source vulnerabilities with greater accuracy so that you can continue leveraging open source libraries without increasing risk. And because 90 percent of the most concerning high-priority flaws have a fix according to SOSS: Open Source Edition, fixing these found flaws after scanning can be an efficient process. So while open source libraries might have a larger than expected attack surface, staying on top of that code with the right scans and an eye for prioritization is key to reducing risk.

The Problem: We're seeing a surplus of high and very high severity flaws in our code but aren't sure where to start.

The Fix: Our data shows that some languages carry more ***high-risk flaws*** than others, which means code written in specific languages should be crafted and tested thoughtfully. For example, 59.3 percent of C++ applications have high and very high severity flaws, along with 52.6 percent of PHP applications. The flaws that more frequently plague PHP include cross-site scripting (XSS), cryptographic issues, directory traversal bugs, and information leakage vulnerabilities. For C++, those common issues include error handling, buffer management errors, numeric errors, and directory traversal flaws, while Java leads with CRLF injection flaws, code quality issues, information leakage, and cryptographic issues.

No matter which language you prefer, understanding the ***flaws that impact them most*** will help you prevent errors before they become bigger headaches. Implementing secure coding practices and utilizing hands-on training to increase know-how will help ensure that the security of your applications can keep up with your development needs regardless of language. When you're empowered to not just find but also fix these high and very high-severity flaws in your code, you're on your way to becoming a more security-savvy developer.



Download volume 11 of our State of Software Security report to learn more about the latest trends in application security, and the steps you can take to improve the security of your code.

Download the Report

VERACODE

Veracode is the leading AppSec partner for creating secure software, reducing the risk of security breach and increasing security and development teams' productivity. As a result, companies using Veracode can move their business, and the world, forward. With its combination of automation, integrations, process, and speed, Veracode helps companies get accurate and reliable results to focus their efforts on fixing, not just finding, potential vulnerabilities.

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