
6
Urban Myths
About Secure Coding
VERACODE

# Urban myths, whether rooted in reality or fabricated entirely, have the power to change perception.

In software development, where the security of your applications relies on best practices and proven methodologies, such urban myths can perpetuate risk by making prevention and remediation seem cumbersome. In turn, common conundrums like securing open source code or properly managing cryptography feel like too much of a hassle.

**WHILE THE MYTHS ARE COMMON, THE REALITY IS CLEAR:**

Taking ownership over your code with the right tools and methodologies means you're creating applications that carry far less risk than ever before.

Read on for six common urban myths about secure coding—and learn practical guidance for how to go about overcoming them.

MYTH


Open source code is more secure because there are "more eyes" on it.

REALITY

## You know that open source code saves valuable time, providing functionality that would be tedious to build from the ground up.

It's also extremely common: according to data from GitHub, 94 percent of active repositories in JavaScript rely on open source, with Ruby, .NET, and PHP hovering around 90 percent as well.

But there's a notion that simply because open source code is used by so many people and so many languages, it must be more secure because more eyes mean better security.

**RIGHT? NOT QUITE.**

It is true that open source code naturally has more eyes on it, but that doesn't mean anyone is keeping the code updated with the latest security information or letting you know that there's a patch available. There's a false sense of security surrounding open source code.

IN REALITY

**It's up to you and other security-minded developers to stay on top of known vulnerabilities in the libraries you use and understand how to fix them with tools like Software Composition Analysis (SCA).**

**We know from Veracode's State of Software Security: Open Source Edition report that 71 percent of applications have a flaw in an open source library upon first scan.**

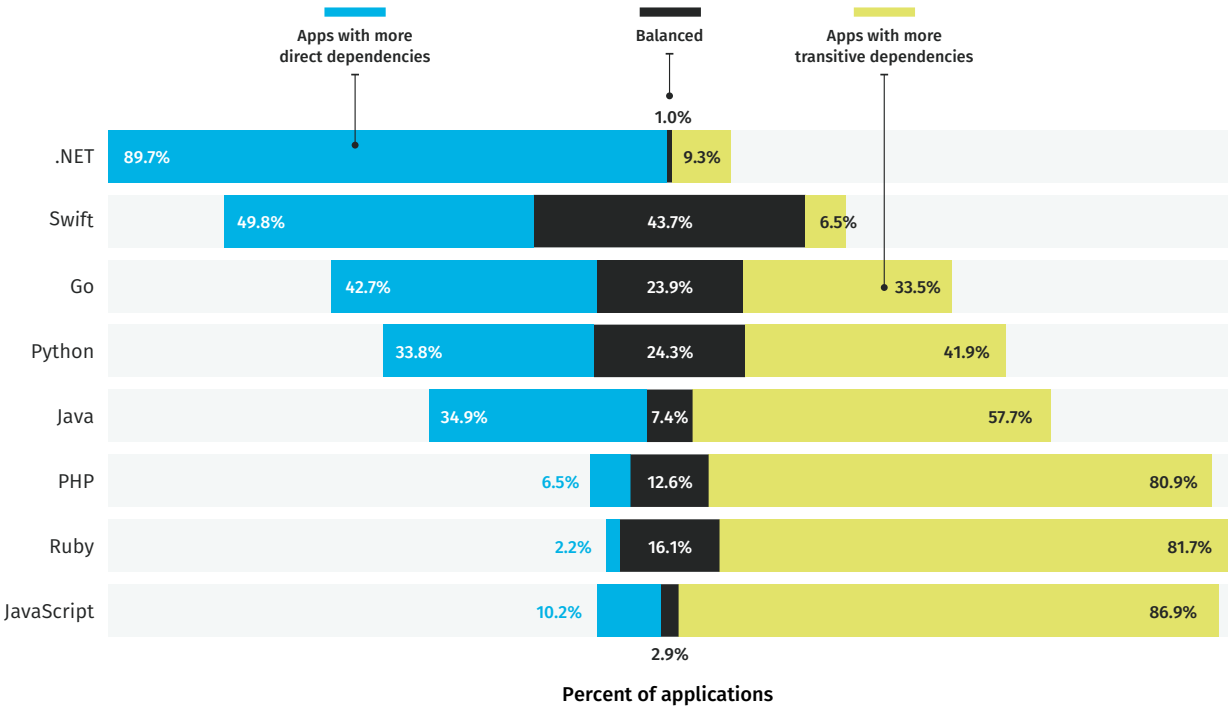
Even more daunting, the most common flaws found in open source libraries are often nasty ones:

```
CROSS-SITE
SCRIPTING
+
INSECURE
DESERIALIZATION
+
ACCESS-CONTROL
FLAWS
=
ABOUT 75 PERCENT OF
VULNERABILITIES IN
OPEN SOURCE LIBRARIES
```

**Additionally, some languages are hit harder by open source vulnerabilities as they carry more dependencies, which means that you must do your due diligence when it comes to scanning third-party code.**

For example, applications written in Ruby, PHP, JavaScript, and Java have more of an attack surface from transitive dependencies — in other words, a library that is dependent on another library — but that doesn't mean you need to shy away from leveraging open source components in these languages.

Instead, manage your libraries closely to keep track of them and their security needs, and implement scans for greater visibility into existing issues so that you can prioritize patches and fixes.



MYTH

# 2

Fixing open source vulnerabilities requires a time-consuming refactoring of code.

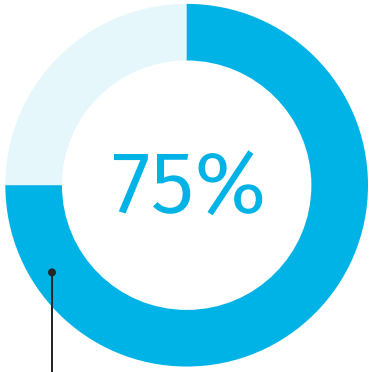
REALITY

**As previously outlined, open source libraries are often riddled with security flaws and vulnerabilities — some of which can turn into dangerous exploits if left unattended.**

---

BUT THERE'S A SILVER LINING

**Most flaws in open source libraries are fixable, and the fixes are minor.**




In fact, almost 75 percent of known vulnerabilities are fixable with a simple library update to patch exploits. As most security flaw fixes are minor updates or even just patch revisions, they generally do not change APIs and thus are not likely to disrupt your work.

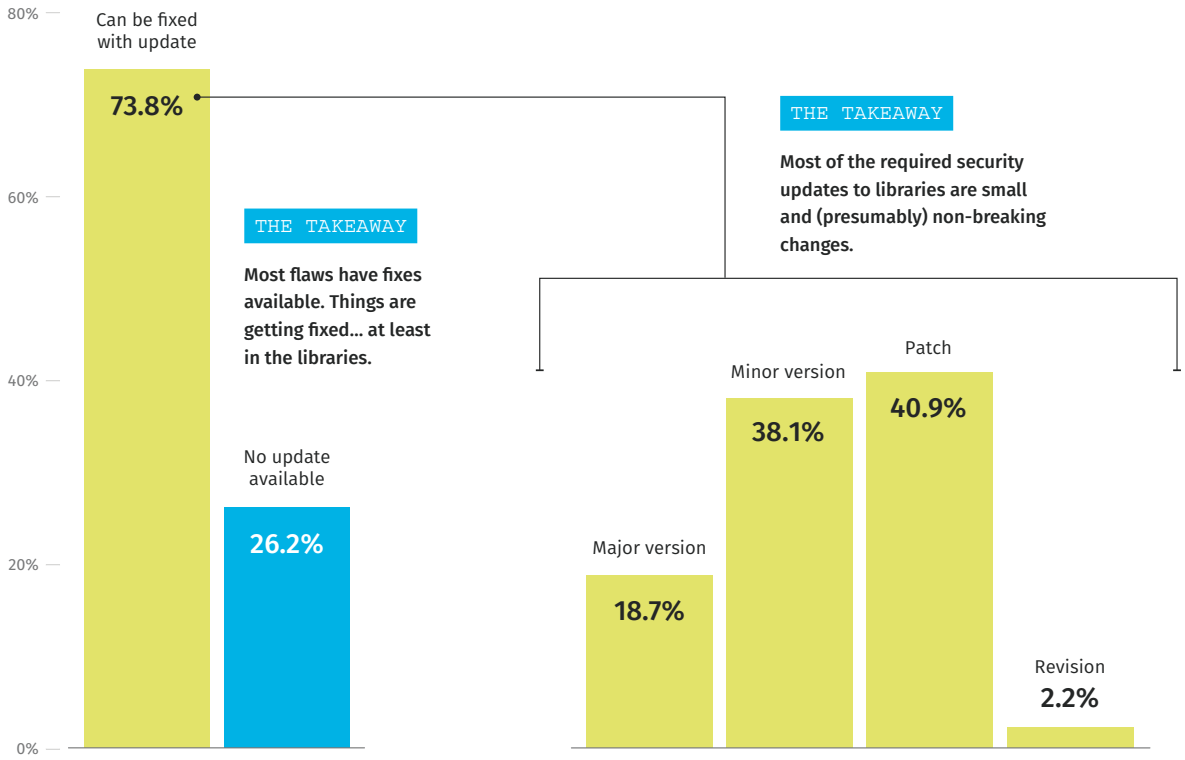
While some flaws that plague open source libraries in particular are scary — broken authentication and cross-site scripting for example — we know that over 90 percent of the highest priority flaws have a fix available.

TO BE EXACT:

**96.4 percent of broken authentication vulnerabilities in open source libraries** *are fixable with a published update*

**90 percent of cross-site scripting and broken access control flaws** *are fixable with a published update*

Opting for scanning tools with features like automated pull requests to generate fixes from GitHub and functions that show you just how likely a fix is to break your app with code changes is a good step towards securing your open source code. While the flaws are daunting, preventing and fixing vulnerabilities in open source code doesn't have to be.




*Percent of library flaws with an available update*

*Types of updates available for library flaws*

# Prioritizing Open Source Flaws

**There's no getting around the various types of dependencies in open source code that can bring risky excess code along for the ride. And while developers often only use a small portion of a library, if the entire library is flagged as vulnerable, the development process is bottlenecked as the team gathers data to decide which flaws seem most severe.**

## DON'T TAKE FLAWS AT FACE VALUE

It's important when prioritizing exploits to not simply lean on scores based on models of exploitability and maturity. With so many other factors, severity scores don't always reflect the true risk a flaw carries and are not very meaningful for prioritization.

# So where do you start when you want to save time (and sanity)?

**Integrating tools like Software Composition Analysis (SCA) provides immediate, actionable guidance to help remediate flaws before they add to your backlog of security debt.**

Once you've identified the risk at hand, your team may decide to take results from an SCA scan and remediate the flaw immediately, mitigate the flaw with a patch to reduce exploitability, or accept the vulnerability with an understanding of the consequences. It helps you automate your work, too, so that you can more accurately predict how much work and rework mitigation might require.

If you decide to take an asset-focused approach that gives highest priority to vulnerabilities associated with critical assets, make sure you're creating tailored requirements for each of your applications with guidance from scanning tools, and not simply basing your decisions on media coverage or emerging "severe" threats. Take into account how long it'll take to fix specific flaws and if there are safer versions of your open source code.

It's also critical to consider if there are potential legal implications from leaving a vulnerability unmediated, examine where the assets live in your environment, and understand if the exploitable vulnerability is in your call path.

When you're ready to tackle your security backlog, examine how particular applications use vulnerable methods and prioritize them in a way that reduces the immediate threat quickly. Many flaws need thoughtful steps that factor in fixes like available updates for patching, a plan of attack for dependencies, and other considerations that relate to library versions.

Once you have the right tools in place, as well as priorities for remediation set based on the needs of your organization and the nature of your applications, understanding which exploits pose an extreme risk (and which simply need a patch) is a much more efficient process.

**Learn more about Veracode Software Composition Analysis**

MYTH

3

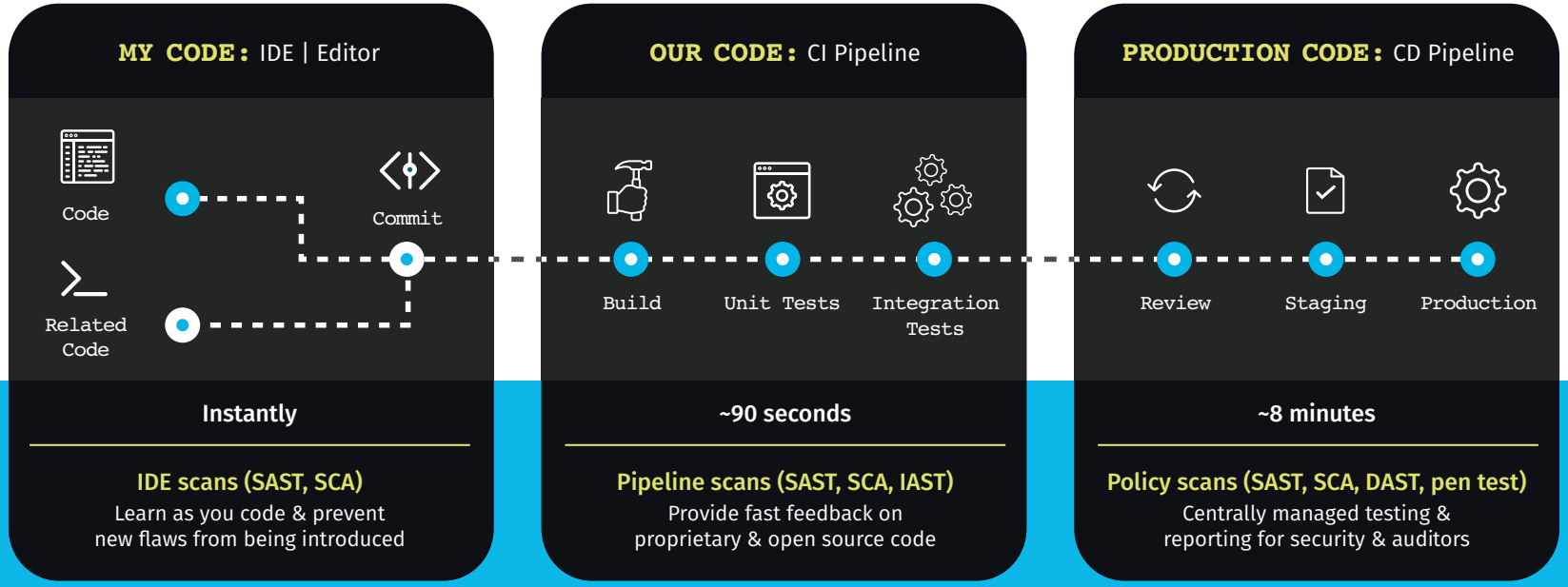
I can trust my favorite developer tools to keep my code secure and give me all the security features I need.

REALITY

## It's easy to treat security as an afterthought, especially when you have security "options" already built into the software development tools you've been using for years.

It can feel like those options are a safety net when in reality they're simply helping you check the most basic of boxes to scan for issues without offering clear data, which means you're likely letting very risky flaws and vulnerabilities slip through the cracks.

These additional security options built into the tools you use to write code aren't necessarily a negative step; it's just that they typically lack the comprehensiveness necessary to actually be effective. Without the right tools in place — meaning solutions embedded strategically to protect you at every step of the development process — you're creating more long-term work without mitigating risk.




**Alternatively, this may mean that only the security team is scanning your application, and doing so right before production, which causes security tech debt to pile up.**

If you're using the wrong scans in the wrong places throughout your SDLC, that means more work (and rework) down the road, and likely unhappy teams that are concerned with security, productivity, and business cost. The solution starts with embedding and automating the right tools in the right places.

Get comfortable with robust scanning solutions that fit into your existing workflows, instead of point solutions that fail when it comes to security. With security plugged into each stage of development from My Code to Our Code and through to Production Code, you're ensuring coverage for your CI/CD pipeline and reducing the stress that comes from mounting security debt. That way you can continue working in an environment you know best while scanning early and often to make more secure applications.

MYTH

# 4

Using more testing types will just lead to more findings and slow everything down, causing unnecessary headaches.

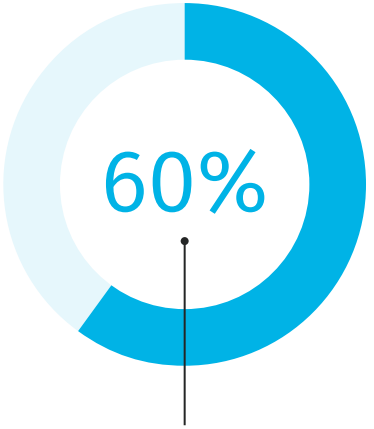
REALITY

**While it might seem like it's adding extra work on the surface, having more than one testing type embedded into your development process saves time as you're able to catch more flaws before the production stage.**

---

That means you won't have to remediate a pile of vulnerabilities later on when it's more of a hassle, and you'll have peace of mind knowing your code is more likely to pass policy checks.

Although many teams decide to only focus on static analysis (SAST), dynamic analysis (DAST) can uncover flaws that are harder for SAST to detect. It finds issues within the environment and not just in the code itself — that's a positive, as we know from *State of Software Security v11* (SOSS v11) that exposing sensitive information through environmental features is a flaw existing in 60 percent of applications.



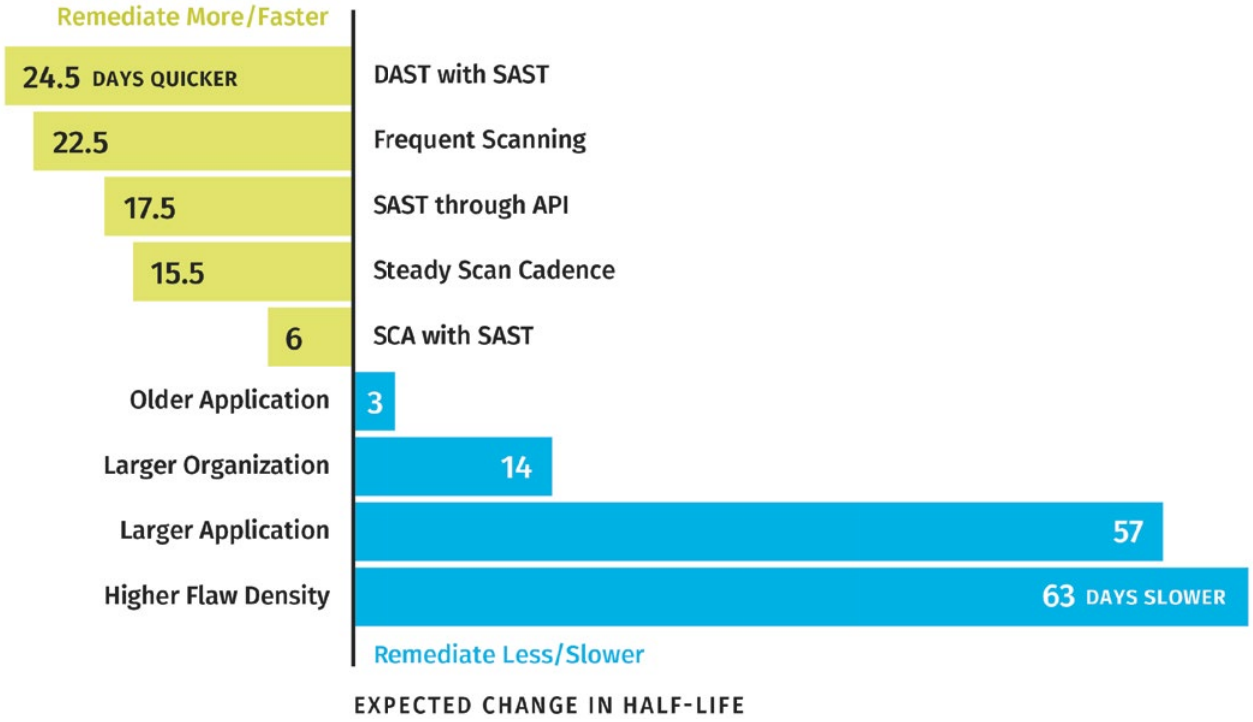

**of applications carry a flaw that exposes sensitive information through environmental features**

**Data from SOSS v11 also shows us that when organizations pair DAST with SAST testing, they are able to remediate 50 percent of security flaws 24.5 days sooner than other organizations that only use one testing type.**

## HERE'S WHY:

Dynamic scanning is crucial to finding issues with server/deployment configuration and authentication, helping you evaluate how the application will interact with its environment to uncover flaws and exploits that may have otherwise gone unnoticed.

Security issues feel a lot more real as DAST sheds light on those otherwise hidden threats; with DAST integrated into your security testing procedures, you have the ability to see just how an attack might unfold at runtime so that you can think more strategically. And while you'll discover a higher number of flaws with DAST as it scans all applications and interconnected structures within an environment, you'll have the ability to remediate them quickly and focus on reducing your risk through prioritization.



MYTH

# 5

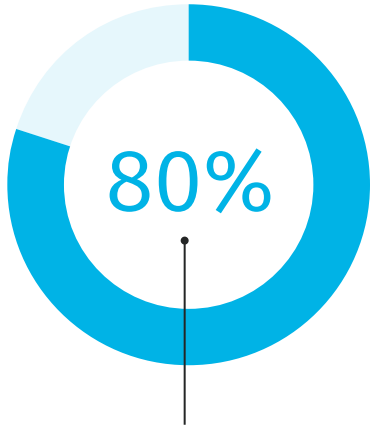
PHP is a "dying language," which means I don't need to worry about understanding the risks.

REALITY

## Although PHP might've been dubbed a "dying language" after its creation in 1994, it is still alive and well.

Despite different schools of thought (some may think it's obsolete or will soon be overshadowed by newer languages), PHP powers nearly 80 percent of all websites built on known programming languages, and the latest version of PHP — version 7 — is used by nearly 60 percent of all websites running PHP.

Some of the Internet's biggest sites like Wikipedia, Etsy, and Facebook were built on PHP. PHP-based publishing platforms like WordPress and Drupal are extremely popular, and they often incorporate sophisticated functionality like e-commerce.




**of all websites built on known programming languages are powered by PHP**

## SO, IF PHP IS HERE TO STAY, WHAT'S THE BAD NEWS?

You can't ignore the risk that it brings. We know from Veracode's SOSS: *Flaw Frequency by Language* report that 53 percent of PHP applications have high and very high-severity flaws, with the most common being cross-site scripting, cryptographic issues, directory traversal vulnerabilities, information leakage flaws, and problems with untrusted initialization.

**When we look at the data in heat map format to compare flaw frequency by language, PHP definitely brings the heat as a big offender:**

| | .Net | C++ | Java | JavaScript | PHP | Python |
|---|---|---|---|---|---|---|
| 1 | Information Leakage 62.8% | Error Handling 66.5% | CRLF Injection 64.4% | Cross-Site Scripting (XSS) 31.5% | Cross-Site Scripting (XSS) 74.6% | Cryptographic Issues 35.0% |
| 2 | Code Quality 53.6% | Buffer Management Errors 46.8% | Code Quality 54.3% | Credentials Management 29.6% | Cryptographic Issues 71.6% | Cross-Site Scripting (XSS) 22.2% |
| 3 | Insufficient Input Validation 48.8% | Numeric Errors 45.8% | Information Leakage 51.9% | CRLF Injection 28.4% | Directory Traversal 64.6% | Directory Traversal 20.6% |
| 4 | Cryptographic Issues 45.9% | Directory Traversal 41.9% | Cryptographic Issues 43.3% | Insufficient Input Validation 25.7% | Information Leakage 63.3% | CRLF Injection 16.4% |
| 5 | Directory Traversal 35.4% | Cryptographic Issues 40.2% | Directory Traversal 30.4% | Information Leakage 22.7% | Untrusted Initialization 61.7% | Insufficient Input Validation 8.3% |
| 6 | CRLF Injection 25.3% | Code Quality 36.6% | Credentials Management 26.5% | Cryptographic Issues 20.9% | Code Injection 48.0% | Information Leakage 8.3% |
| 7 | Cross-Site Scripting (XSS) 24.0% | Buffer Overflow 35.3% | Cross-Site Scripting (XSS) 25.2% | Authentication Issues 14.9% | Encapsulation 48.0% | Server Configuration 8.1% |
| 8 | Credentials Management 19.9% | Race Conditions 30.2% | Insufficient Input Validation 25.2% | Directory Traversal 11.5% | Command or Argument Injection 45.4% | Credentials Management 7.2% |
| 9 | SQL Injection 12.7% | Potential Backdoor 25.0% | Encapsulation 18.1% | Code Quality 7.6% | Credentials Management 44.3% | Dangerous Functions 6.9% |
| 10 | Encapsulation 12.4% | Untrusted Initialization 22.4% | API Abuse 16.2% | Authorization Issues 4.0% | Code Quality 40.3% | Authorization Issues 6.8% |

**Despite being prone to risky vulnerabilities, PHP remains popular with developers and it likely isn't going anywhere anytime soon.**

### THAT'S BECAUSE PHP IS:

- Easy for beginners to learn, which means developers with some basic skills can pick it up quickly and start coding websites right away.
- Already popular within the broader developer community, with an established base of developers like you who bring a wide range of experience.
- Streamlined to work well with open source code, enabling developers to complete coding projects in a more timely manner.
- Ready to go with fundamental features like object orientation, packet management, and OS compatibility to facilitate modern web design needs.

---

**While PHP certainly isn't dying anytime soon, it's critical that you stay on top of the risks and exploits that it can bring — especially when leveraging open source components.**

MYTH

# 6

Cryptography is too hard to implement, so we can just leave it up to the security team.

REALITY

**Just like any other programming concept, learning the basics of cryptography without needing the skills of an expert mathematician or a cryptographer is possible.**

---

You simply need to familiarize yourself with cryptographic concepts, libraries, and best practices.

CRYPTOGRAPHY...

- Establishes Confidentiality, Integrity, and Availability (CIA) in safeguarding data and communications, which is critical to the security of every organization.
- Shows up everywhere in our daily lives, from Wi-Fi security to your phone's lock function and even key fobs for cars. Due to the criticality and complexity of any crypto-system, you need to, at a minimum, make sure your security is audited.
- Is not going to protect you against all security mishaps, though. You still need to consider common application security issues like input validation, authentication, and authorization, and provide relevant mitigations.

# Some cryptographic best practices include:

### DON'T ROLL YOUR OWN CRYPTO

A mere 17 percent of bugs come from crypto libraries according to a study from MIT CSAIL, and the rest are misuses or from developers rolling their own crypto.

### MOST CRYPTO PRIMITIVES NEED A RANDOM NUMBER GENERATOR (RNG)

Make sure your RNGs are cryptographically secure, which means it takes its source of entropy from underlying OS, such as /dev/urandom.

### FOR ENCRYPTION

Use only Authenticated Symmetric encryption, such as AES-GCM or ChaCha20-Poly1305 schemes.

### FOR CRYPTO-SYSTEMS NEEDING HASHES

Employ SHA2/3 family of hashes — not any algorithm from the MD family.

### WHEN YOU NEED MESSAGE AUTHENTICATION CODES (MAC)

Use HMAC (or hash-based message authentication code) with any of SHA2/3 family of hashes.

### DON'T STORE PASSWORDS OR SENSITIVE INFORMATION IN CLEARTEXT

Instead use key derivation functions (KDFs) for storing passwords, such as Argon2 or Scrypt.

### CONSIDER USING ELLIPTIC CURVES

(Edward Curves or NIST-approved curves) for any public key applications, such as digital signatures.

### POOR KEY MANAGEMENT IS ONE OF THE MOST COMMON CRYPTOGRAPHIC FAILURES

Use a mature Key Management System (KMS), such as cloud-provided options from Amazon (AWS), Google (GCP), or Microsoft (Azure). For applications that demand the highest level of security, consider using Hardware Security Modules (HSM).

**For more specific guidance and recommendations, read this Crypto series.**

# Code More Securely with Veracode Security Labs

**Now that you know six of the most common urban myths and their realities, where should you start?**

Sharpening your secure coding skills through real-world training not only helps you prepare to catch and remediate flaws on the fly, but also trains you to write better code and prevent exploits altogether.

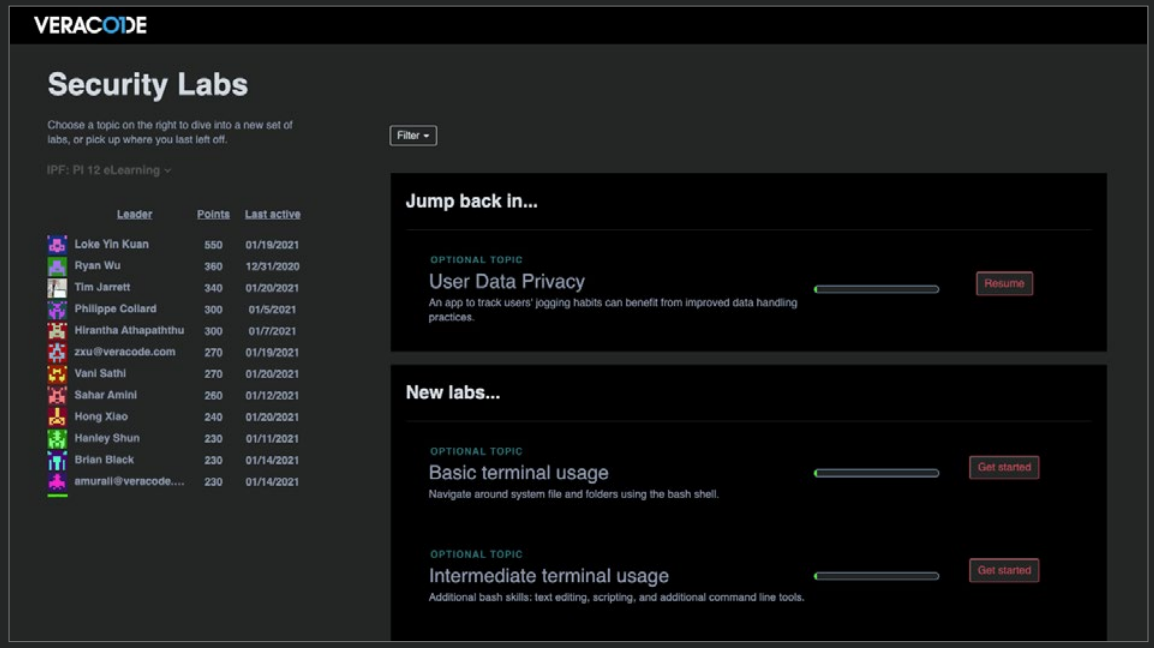
Veracode Security Labs is a hands-on training platform where you can exploit and patch real applications in contained environments to see how threat actors operate. That gets (and keeps) you one step ahead. In these interactive labs, you're guided through with automatic progress checks and additional practice challenges to reinforce acquired skills.

## YOU'LL LEARN:

- What not to do in the future when writing code
- How to quickly patch found flaws so that you can keep your projects moving forward
- How to prevent and remediate common issues in code quality, from injection flaws to cross-site scripting exploits, data privacy issues, and modern application weaknesses

## EVEN BETTER:

Veracode Security Labs Community Edition is a complimentary version of the platform with courses ranging from beginner to advanced. Hit the ground running with these hands-on labs, gauging your progress in real time to keep track of your improvements in secure coding.




**Sign up now and start cultivating your security skills**

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.

Veracode serves more than 2,500 customers worldwide across a wide range of industries. The Veracode cloud platform has assessed more than 14 trillion lines of code and helped companies fix more than 46 million security flaws.

**www.veracode.com** **Veracode Blog** **Twitter**