VERACODE

HOW DEEP RISK ANALYSIS BEATS FAST SCANNING:

The Hidden Cost of Surface-Level Code Security

INTRODUCTION:

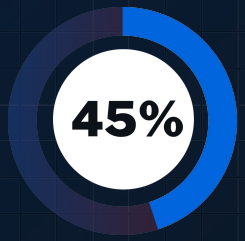
False positives from security scanners cost one enterprise over 200 developer hours in a single quarter. At a loaded cost of $150/hour, that's $30,000 in wasted productivity. Frustrated, they disabled their scanners entirely. Multiplied across dozens of teams, this problem costs enterprise organizations millions, and it is not an isolated issue. This impossible trade-off between noise and risk is why organizations need a more intelligent approach to security.

Traditional code security solutions force an impossible choice: drown in false positives or miss critical vulnerabilities. Veracode offers a superior approach, drastically reducing false positives to less than 1% (compared to industry averages of 5-30%) while maintaining leading detection rates, all integrated seamlessly into existing workflows to enable fast identification and remediation of risk.

The urgency for effective security has never been greater. AI coding assistants are accelerating development velocity tenfold, but they are also introducing vulnerabilities at the same rate. **Veracode's 2025 GenAI Code Security report indicates that 45% of AI-generated code failed security tests for various OWASP Top 10 vulnerabilities.** The "old security gates" are obsolete, and "guardrails" allow too many flaws to slip through. These guardrails, which give developers the option to disable security scanners, often lead to developers abandoning secure development practices after countless hours wasted on false positives (or worse, after your 'fast' scanner missed the vulnerability that led to last quarter's breach).

There's a better way.

The problem isn't with security scanning itself, but with the false dilemma it presents between speed and accuracy, between shipping features and shipping securely. This paper will demonstrate why traditional approaches are failing and introduce a proven alternative: continuous, efficient, deep security analysis that developers will actually embrace.



AI-generated code failed security tests for various OWASP Top 10 vulnerabilities.

KEY TAKEAWAYS:

- **The $120K Problem:** Why teams waste $120,000 annually per team on false positives and often disable code scanners entirely
- **The 1% Solution:** How Veracode achieves <1% false positives (vs 5-30% industry average) through deep analysis, not pattern matching
- **85% Faster Fixes:** Why Continuous scanning delivers fixes in seconds or minutes instead of hours – without blocking deployments
- **$2M + ROI:** The documented return from eliminating exposure windows and preventing breach and compliance risks.

PART 1:

The False Positive Crisis

One customer turned to Veracode when false positives became a critical pain point, burning through 200+ developer hours in a single quarter and destroying their faith in prior security scanning tools. This "cry wolf" scenario is an industry-wide crisis, costing businesses millions in wasted effort and leaving crucial vulnerabilities exposed.

A recent academic study found that, of 240 academic papers about static application security testing tools, fully two-thirds of the papers flagged high false positive rates as an impediment to use, and identified the economic cost of false positives:

> "When a developer has to spend hours tracking down a reported issue only to find that it wasn't an issue, it can have a cascading result. This wasted time can lead to a decreased trust in the tool, general frustration, and overall reduced productivity."

In a compelling real-world scenario, a customer in the banking industry challenged us to understand why a competing static analysis tool found hundreds more findings than Veracode did. Our security research team dedicated 75 research hours to investigate the reported findings and found that every single finding was a false positive. Those 75 hours for our research time might have translated to hundreds of hours wasted for our customer's developers.

Similarly, a European bank struggled to operationalize a high-FP static solution for over a year and was never able to bring it to production – tuning out the false positives was just too much work. Simply put: after too many false positives, developers tend to turn the tools off.

This is a worst-case scenario for many security teams. Instead of getting some static analysis findings fixed, their development teams are instead blind to any risk being introduced, thanks to the persistent noise levels from poorly tuned tools.

Veracode has long had a reputation for some of the lowest false positive rates on the market. Our commitment to high-quality, low-noise results is supported by several key capabilities:

Reachability and “taint tracing”

A flawed programming construct that is never executed or that an attacker cannot invoke and control is no security problem at all. Veracode ensures that we report only reachable flaws by performing dead code elimination, not analyzing code in comments, understanding data flow (how data moves through the program) and control flow (how the program operates at runtime), and especially tracing the flow of potentially tainted data through the program using pointer analysis to establish whether an attacker can get a payload to the part of the code in question. That’s way more sophisticated than just grepping for a pattern in the code (i.e., simple pattern matching using regular expressions).

Full program analysis

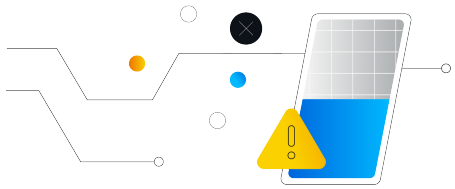
While one way of reducing scan time is to look only at a small piece of code, like a single class or single file, this has tremendous impacts on both false positives and false negatives. By examining only the class, we lose all information about how it’s invoked at runtime, and can no longer perform reachability, data flow analysis, or reliably determine whether an attacker can get there. Veracode supports full program analysis on up to 5GB of code at a time — a benefit when you’re considering very old codebases — or on collections of microservices.

Framework support

One of the challenges of performing static analysis on a modern application is that much of the control flow logic of the application is not implemented within the application code itself. In most modern web applications, the actual control flow logic is implemented in a framework that implements a standard control flow pattern (like model-view-controller), and the application only contains instructions to the framework on how to operate. This means that a static scanner that doesn’t understand the framework may see a single method call into the framework, a series of entries in a configuration file, or annotations in a Java class, and miss the fact that the framework is being called unsafely. To handle this situation, the static scanner must fully support the data and control flow of the framework, including understanding what the different methods of the framework do, and be able to model it so that the customer’s code can be evaluated in context. That’s one of the reasons that Veracode’s support for hundreds of frameworks across all our supported languages is important.

To increasingly perform the critical task of evaluating code in context, Veracode analyzes the code our customers scan to understand the relative importance of supporting new frameworks, which are added regularly.

Language	Frameworks
Java	63
.NET	33
JavaScript	22
Android	3 (plus Java and Kotlin frameworks)
Apex	3
Cross-Platform Mobile App Toolkits	7
Supported COBOL dialects and standards	17
Go	4
PHP	4
Python	9
Scala	3



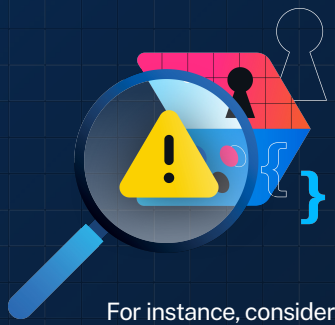
While competitors use simple pattern matching, Veracode understands the actual control flow of 170+ frameworks across 11 languages. This context awareness is why our false positive rate is less than 1% while competitors range from 5-30%.

Security-sensitive context filtering:

In some cases, particularly with vulnerability types related to numeric size or behavior, it can be hard to identify flaws based solely on a scan that relies on data and control flow. There are often security context factors that mean that a potential numeric flaw is a coding error but could never have any security implication. Veracode has implemented security sensitive context (SSC) rules to review these findings and suppress any that occur in a security-irrelevant case, thus resulting in a dramatically lower false positive rate for the scan.

Veracode is Regularly Adding New Framework Support

TECHNICAL DEEP DIVE ON SSCS:



For instance, consider a number being passed into a function that has no security consequences for receiving a negative value:

```
int main(void)
{
  unsigned int u = 0;
  u--;
  printf("%d\n", u);
  return 0;
}
```

Without the security sensitive context filtering, we would report an integer overflow flaw; with the rule, this flaw is suppressed from the report. Veracode uses SSCs on a variety of numeric and memory allocation flaw categories.

Crosscheck

Given the forking nature of paths within a software application, there may be many ways an attacker can reach and exploit a flawed piece of code. It's not enough to find one of the paths to establish whether the code is vulnerable; you need to look at all of them. Veracode's crosscheck process finds and reports on all the code paths that allow an attacker to reach the vulnerable code via a patented process (US 9,286,063).

Continuous customer feedback loop

Every time a customer raises a question about static results, it's reviewed and escalated as needed to our engineering team. Fixes are validated against a testbed of thousands of real-world and synthetic applications, and when released the fixes improve scan results for each customer thereafter. This continuous feedback loop is pivotal to how Veracode has built the lowest false positive rate in the business. With thousands of organizations relying on our static analysis engine - from Fortune 500 banks to cutting-edge startups - this continuous feedback loop has refined our accuracy over millions of scans and hundreds of trillions of lines of code.

PART 2:

The False Negative Danger

So, if false positives are bad, surely the right thing to do is to optimize completely for low FPs, right? Not so fast. As with so much that matters in life, the balance — between noise on the one hand and critical issues on the other — is the important thing. Missing a critical finding can result in high financial and reputational costs.

However, each business has a trade-off in terms of developer time and revenue impact; making those trade-offs is a risk-based decision. And security teams should be asking themselves what the cost of a potential missed finding could be. One security practitioner in the automotive industry responded in a 2024 study of industry perspectives on static analysis testing,

> "I just told you the amount of the price of the bug (in millions)... False negative-that one is going to kill you."

Why does this matter when planning an application risk management strategy? The challenge for security and DevOps teams is that while you can find a scanner that doesn't produce a lot of false positives, or find a scanner that is thorough, finding one that does both is the challenge. Fortunately, some of the same technologies that help Veracode with false positives help keep false negatives at bay, including data and control flow modeling, full program analysis, rich frameworks support, and our continuous customer feedback loop.

Rich frameworks support is especially critical here; since most modern software development frameworks provide methods for database access, display templating, APIs and control flow, understanding these framework methods is required for finding cases where a software program uses them incorrectly, introducing a security issue.

But there are challenges with avoiding false negatives, too. Doing proper interprocedural modeling, data and control flow modeling, and full program analysis are computationally expensive. And an organization's "crown jewel" applications are frequently not small, well architected microservices that scan and publish quickly; they're the gnarliest, huge monoliths that defy all efforts to decompose or retire them, meaning their scan times can be lengthy.

WHY OTHER VENDORS PUSH SPEED OVER ACCURACY

The dirty secret of 'fast' scanners: they're only fast because they skip the hard work. So-called “fast” SAST scanners from Snyk, Checkmarx, and GitHub Advanced Security cut corners by skipping full program analysis. This shortcut results in 5–30x more false positives, trading true accuracy for the illusion of speed and leaving you vulnerable.

It’s speed without substance — like a spell-check that only scans first letters. When Snyk, Checkmarx, or GitHub Advanced Security tout sub-minute scan times, ask them: 'Are you doing full program analysis? Do you understand my framework's control flow? Are you tracing every possible attack path?' The answer is no.

Fast scanners make great demos. They can scan your hello-world app in seconds. But they achieve this speed by cutting corners: pattern matching instead of data flow analysis, partial scans instead of full program analysis, and ignoring framework context. When you're protecting production systems, would you rather have a fast wrong answer or a thorough right one?

Feature	Traditional Simple SAST	Veracode Deep Analysis
Analysis method	Pattern matching	Full data flow analysis
Scan scope	Partial/file scanning	Complete program analysis
Rule application	Generic rules	Framework-aware detection
False positive rate	5-30%	1.1%
Bug detection	Misses complex bugs	Catches sophisticated attacks
Outcome	Fast but shallow	Thorough and accurate

PART 3:

Why Traditional “Shift Left” Fails

At this point, it’s almost an article of faith among security practitioners that you should test as early as possible in the development process. “Shifting left” gives development teams an early indication of where they’ll have to do additional work in the process of releasing a feature to clear security hurdles. The practice can be viewed as the security equivalent of the “spike” story that gives an agile team more knowledge and reduces the uncertainty around the requirements to deliver the feature.

The main benefit of “shift left” is clear: fix bugs when they’re cheaper to fix. But this economic argument completely falls apart when your scanner produces false positives. As we’ve established, false positives aren’t bugs… they’re productivity destroyers. Shifting left with a high-FP tool doesn’t save money; it just wastes developer time sooner. The real benefit of shift left only emerges when development teams get accurate, actionable findings early. This means information shouldn’t block their pipelines, shouldn’t flood them with noise in the name of speed, and should surface the most important findings – including those that require full program analysis and deeper scanning – before software reaches delivery.

This points up a conundrum in “shift left”: there’s pressure to make scanning as fast as possible so that it can gate pipelines, but also to lower noise. There’s little or no back pressure in the system to ensure that false negatives stay low, too. As a result, developers have every incentive to choose fast, shallow scanners that don’t look for many categories of findings, so that they don’t have to deal with many different results.

BREAKING PIPELINES IS HARMFUL

What's worse:
shipping insecure code or not being able to ship a fix for some security problems?

This is a dilemma that confronts security practitioners and development teams alike, and it's not a hypothetical problem. Consider the case of a large legacy (i.e. not created in the last year; no one from the original software team still works on the code) application. Just as most legacy applications have tech debt, almost all of them have security debt as well. And most have not just a handful of flaws, but hundreds or thousands. No software development team on earth is going to fix that much code all in one shot and release it all at once. The smart thing to do is to release each fix in a small bugfix release so you can isolate the testing for each one, minimize the chance that the security fix introduces other problems, and generally ship fixes sooner.

But if you're breaking pipelines on security debt, this strategy requires lots of manual work to close or approve exceptions to the findings, so you can ship out the fixes for the ones you've worked on. Also, testing during a delivery pipeline takes time, which can be particularly painful if the pipeline is running to deliver a critical fix. In some cases, it can mean a longer exposure time for a vulnerability, or a longer mean time to repair, meaning that your system is at risk of attack or is disappointing customers for a longer period of time.

Here's a contrarian approach to the problem of breaking delivery pipelines with security testing results: don't. Instead, we recommend embracing "fail forward" as a key development and security principle. This ensures code changes can safely get to production quickly, rather than waiting for perfection each time. That means not gating scans but ensuring that every code change on every branch gets a high-quality, low-noise security scan, and that those scan results are visible both to the development and security teams.

What are some “fail forward” practices?

Examples include:

- **Make security issues visible and backlogged but fail the pipeline only in the most extreme cases** – For instance, an easily discoverable vulnerability in an open-source library that has readily available exploits with high impact should stop the pipeline, since the expected window of time before an attacker finds and exploits would be short. Doing this in practice requires robust policy controls that allow specifying breaking a build on reachability, not just severity.
- **Maintain pipeline integrity** – Check for security scans performed earlier in the pipeline.
- **Scan alongside, not in line** – Run scans in parallel with CI/CD pipelines to keep delivery velocity high.
- **Accountability** – Set remediation deadlines for all discovered findings based on severity and reachability.
- **Continuous fixing** – Configure the SCM environment to open pull requests containing recommended fixes for issues in a branch, or to comment on an open pull request with recommended fixes for the changes in the pull request. (Veracode Fix can help with both.) This allows development teams to review suggested fixes rather than research issues from tickets.

Once applications are in compliance, then implement “break the build” and other controls to keep them that way.

To make “fail forward security” work

Here are other recommendations:

- Scan all code changes in lower branches to give teams visibility in their workspaces.
- Apply gates on merges/pull requests to the main delivery branch. Development teams that practice peer reviews, change controls and other best practices will typically use this point as a sign-off for other processes. Take advantage of this existing gate to align security controls with development practices.
- Ensure that security issues are placed in the teams’ backlogs.
- Ensure in the delivery pipeline that merge/pull requests were signed off, including security requirements.
- Use automated remediation tools in the pull request process, ensuring security fixes are suggested before code reaches the main branch.

An example of how “fail forward” looks in practice

One of our healthcare customers discovered a critical authentication bypass in a massive monolithic application on a Friday afternoon. Under their previous gate-based Code Security approach, the security scan alone would have added 3 hours to their emergency deployment. By implementing fail-forward principles with continuous background scanning, they deployed the fix in 47 minutes and avoided a potential weekend-long exposure window.

TRADITIONAL GATE-BASED APPROACH:

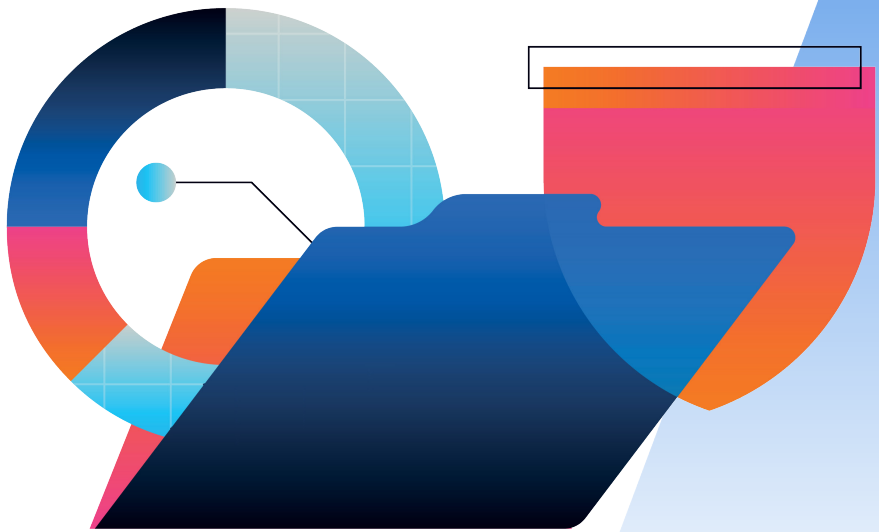
- Full scan required: 3+ hours
- Manual exception process: 30-60 minutes
- Total delay for critical fix: 4+ hours
- Weekend on-call team frustration: Immeasurable

FAIL-FORWARD APPROACH:

- Fix deployed immediately
- Security validation runs in parallel
- Any new issues tracked for next sprint
- MTTR improvement: 85% faster

THE ROI OF FAIL-FORWARD

Consider a typical enterprise that fixes 500 critical findings per year. At 4 hours saved per fix, that's 200 hours of reduced exposure window. For a company with $10M in hourly revenue risk during outages, this approach can save $2M in potential exposure –not counting the improved developer satisfaction and reduced burnout from smoother emergency deployments. And this is a conservative case; some of our customers fix over 50,000 critical findings per year.



SHIFTING LEFT, TO DEVELOPERS AND BEYOND

What about shifting security even further left, to developers? There are strong arguments to be made for putting testing tools in developer environments, starting with IDEs. The value for motivated developers:

- **Check in clean code:** Make sure that you aren't introducing security issues with new features or other code changes by scanning before you push to the repo.
- **Find and fix bugs while you're still working on the code:** Context switching is a productivity killer. Fixing security bugs while the code is already in your head is a great time saver, especially when using AI-driven remediation suggestions.
- **Getting smart about security:** By showing developers how to fix a finding, Veracode's tools empower the developer to get smarter about strategies for reducing software risk, meaning the next time they encounter a flaw, they'll have a leg up.

You can also shift even *further* left. Consider malicious open-source packages: for some packages that execute malicious payloads upon installation, finding them as part of a software scan after they've been checked into the source code repo is too late. Veracode Package Firewall helps secure the earliest stages of development by preventing malicious open-source packages, as well as other highly risky packages, from ever being downloaded in the developer or build environment in the first place.

PART 4:

The Solution: Continuous Repository Scanning

The Clock Is Ticking

Every day you delay moving to continuous scanning is another day of:

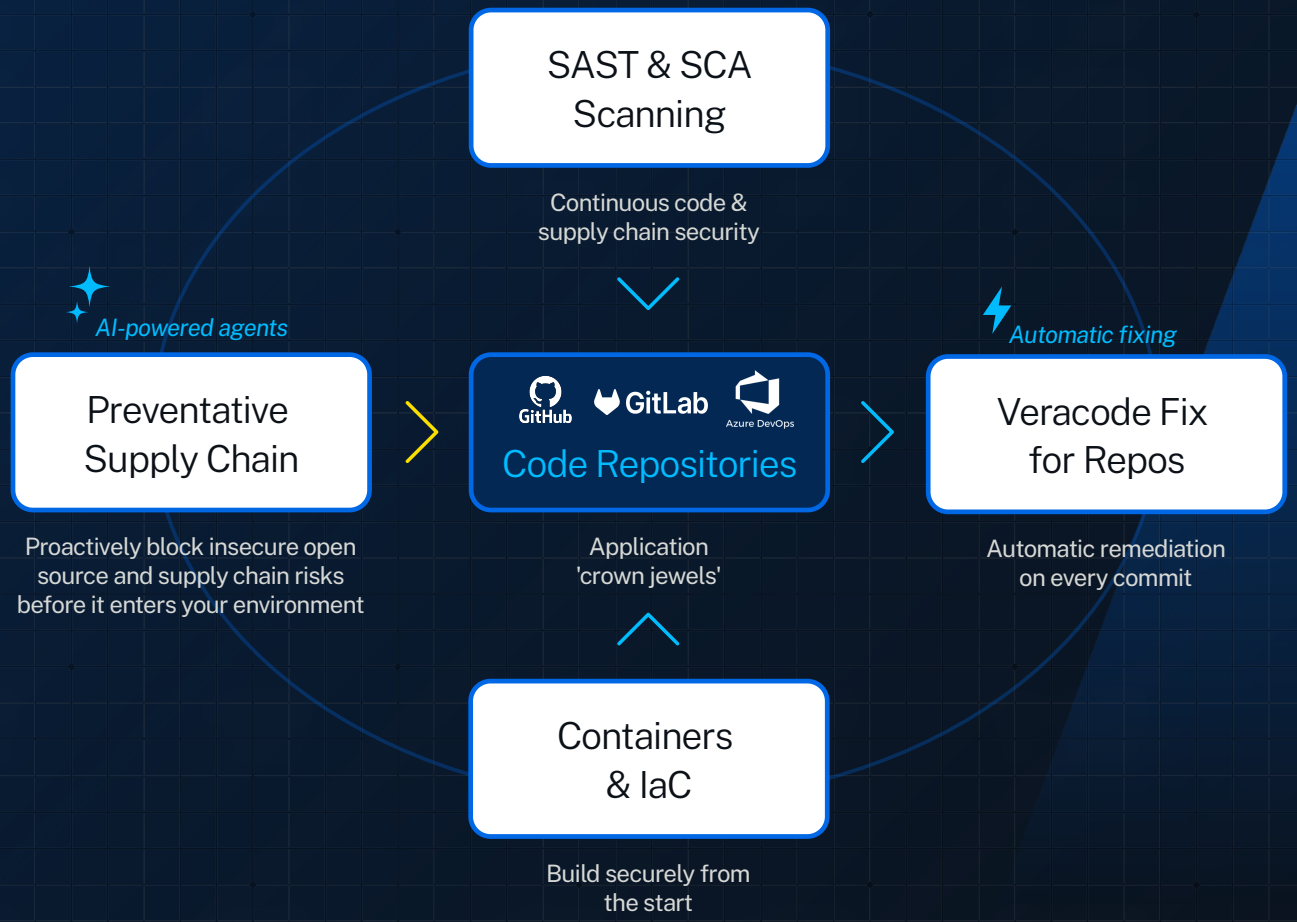
- Developers disabling security tools
- Critical vulnerabilities hiding in production
- Technical security debt compounding

The good news?

You can start transforming your security posture in just 5 minutes.

THE FOUNDATION FOR APPLICATION RISK MANAGEMENT

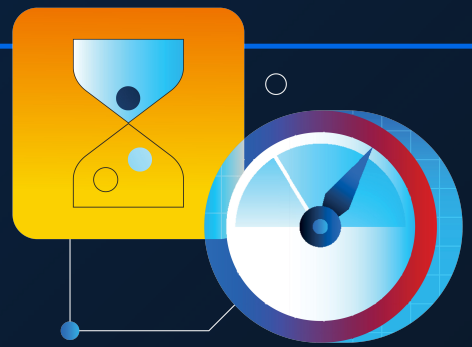
Continuous scanning • Automatic fixing • AI-powered agents • Pull request generation



The last thing that must be said is that, often, getting these workflows right is hard, requiring lots of time and thought from skilled DevOps professionals. But the temptation to customize workflows for each branch, and the conflicting needs between security and development, mean that progress can be slow. And many security tools require special implementation in each pipeline. This is a real problem when businesses have hundreds, thousands, or uncountable multitudes of code repositories with pipelines.

Instead of custom pipeline development, a better way might be to orchestrate security scanning in response to changes in the software repository. That is, on every check-in and merge request via centralized workflows that embed some of the best practices we've discussed above.

While other vendors tried to make their scanners faster (and less accurate), Veracode took a different path. We optimized for accuracy first, then solved the speed problem through architecture: continuous background scanning that never blocks deployments. This isn't a compromise; it's a fundamentally better approach that gives you both depth and speed.



Already trusted by thousands of development teams worldwide, this approach is critical for solving the problems we've discussed above:

- **Low false positives and false negatives:** Deep analysis takes longer than superficial pattern matching, but when it runs continuously in the background without blocking your deployments, that thoroughness becomes a strength, not a weakness. You get enterprise-grade security without enterprise-grade headaches.
- **Early testing:** Opinionated workflows optimize scanning in lower branches for speed while still finding critical problems.
- **Supports "fail forward" and merge gates:** It even gives you the option to block pipelines if you really want to.
- **Issue visibility:** Supports creating issues in the SCM native view of choice for each platform.
- **Security team visibility:** Provides security teams with an overview of the testing workflows and dashboards, reports, and workflow approval ability for security findings.

Additionally, the central configuration saves time and effort for both security and DevOps and allows scaling a security program across all the software in the organization.

The good news is that Veracode's repo scanning workflow tools provide exactly this type of orchestration for the three most commonly used source code management platforms: GitHub, Azure DevOps, and GitLab. And this accuracy-first approach extends beyond SAST. The same continuous scanning architecture handles Software Composition Analysis (SCA) for open source risks, Container Security, and Infrastructure as Code scanning - all with the same low false positive rate that makes our SAST trusted by tens of thousands of teams

Ready to Transform Your Application Security?

Join thousands of teams using deep continuous scanning to secure their code without slowing down.

LEARN FROM PEERS:

See how enterprises reduced security debt by 70%

Read Case Studies

VERACODE

No pipeline changes. No broken builds. Just better security.

SEE HOW IT WORKS:

Get a personalized demo showing continuous scanning on YOUR code

Schedule 30-Min Demo

Contact Sales