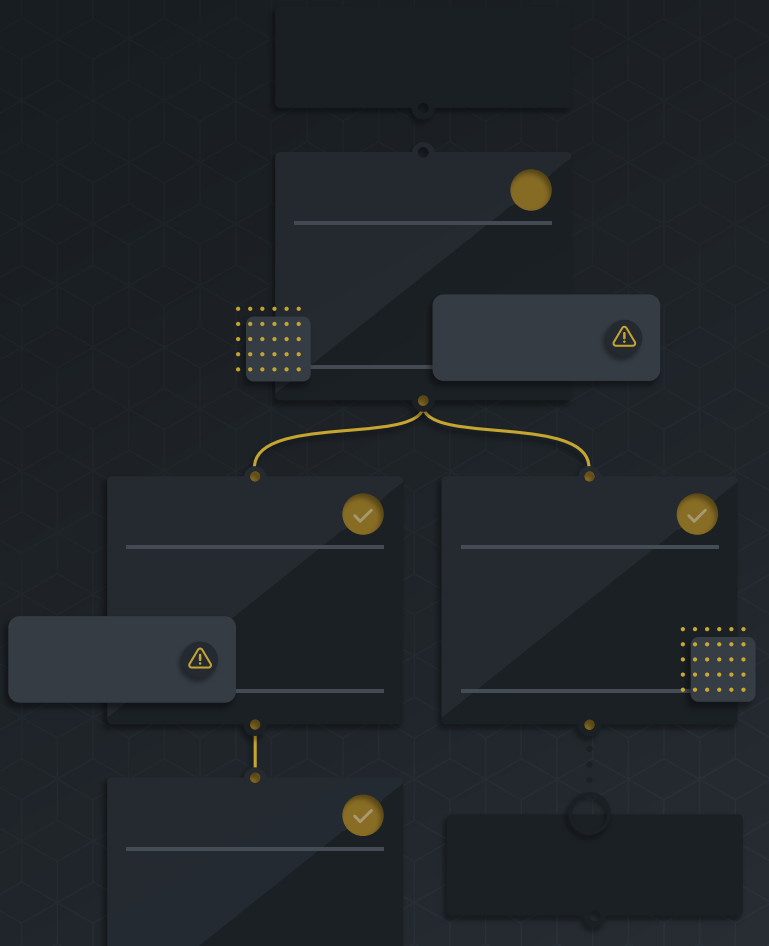
The complete guide to developer-first application security

WRITTEN BY GITHUB WITH ♥

Contents

Introduction



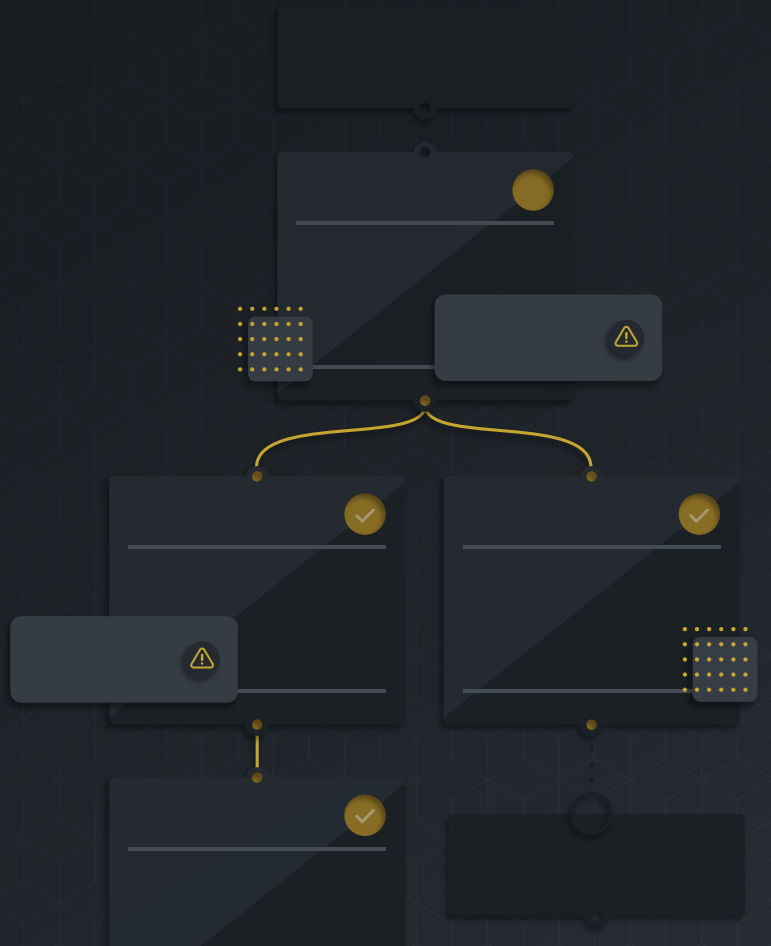
As a result of globalization and digital transformation, business now runs on ones and zeros. No matter the industry, high-performing organizations all compete for the same advantage: Transforming the customer experience into a digital-first medium that stands out.

Since applications fuel these digital experiences, developing applications needed to deliver business processes has become a core competency for organizations of all sizes; every company is now a technology company. At the same time, enterprise applications' increasing use and importance create a prime target for malicious actors—resulting in devastating data breaches. While it can be difficult to pinpoint the initial attack vector for breaches, in retrospect, many of the biggest recent breaches are known to have leveraged vulnerabilities at the application layer.[1]

Given how critical applications are to many businesses—both in terms of the functions they provide and the data they process—why do we keep experiencing application security breaches? Despite an emphasis on application development and improved application security, application vulnerabilities continue to grow linearly with lines of code. How can we break this relationship in order to deliver more secure applications? In this ebook, we'll take a look at the current state of application security and recommend sustainable solutions. We'll also share GitHub's responsibility in securing the world's software, and how GitHub helps organizations deliver more secure applications and empower innovation.

1: 2020 Open Source Security and Risk Analysis Report, Synopsys

Executive summary



Part one: The current state of application security

Application security leverages a system of tools, processes, and best practices to manage application-related business risk. Depending on risk appetite and the criticality of applications, as well as security program maturity, application security can range from simple risk awareness to a well-established pipeline that quickly identifies and remediates vulnerabilities, ideally pre-production. Modern software is built on open source, but as the adoption of open source components increases, so can security risks for both developers and security teams.

For the average organization today, application security consists of a small set of testing tools integrated with the software development cycle. Common current concepts include static application security testing (SAST), dynamic application security testing (DAST), passive and active integrated application security testing (IAST), runtime application security protection (RASP), fuzzing, software composition analysis (SCA), penetration testing, and bug bounties.

> Depending on an organization's maturity level, tooling, and capabilities, application security is either treated as the final gate before deploying an application, or as a series of tests integrated with the development cycle.

Part two: Traditional vs. end-to-end security

Traditional approach: Security as a gate

Having security as a gate prior to deployment is the most traditional approach, and often the first step for organizations just starting with application security. This approach consists of security tests that run during the quality assurance phase. These tests are provided by security teams or third-party vendors, and the outcomes are delivered in bulk to developers for remediation with the expectation that everything will be fixed prior to deploying to production.

In this traditional gate approach, SAST, DAST, IAST, and SCA are the most commonly observed security evaluation tools. Although having security as a gate is better than having no application security at all, this approach causes developer friction and delays in delivering secure applications. Late security feedback causes confusion, manual reviews lead to bottlenecks, and scan results have a high noise-to-signal ratio—all of which lead to developer frustration and disrupt developer velocity.

End-to-end approach: Security integrated into every step of the development cycle

Organizations that are more mature in application security employ an end-to-end approach. This delivers superior results to the traditional approach by providing developers with feedback on their application's security earlier ("shifting security left"), and

leveraging integration and automation capabilities throughout the development lifecycle. However, like the shortcomings of the traditional approach, the end-to-end approach has four main friction points:

1. Integrations require constant upkeep and frequently break due with version updates.

2. Security teams and development teams still work in silos.

3. Automated tools don't solve the problem of false positives.

4. Traditional tools fail to keep up with the pace of the software ecosystem.

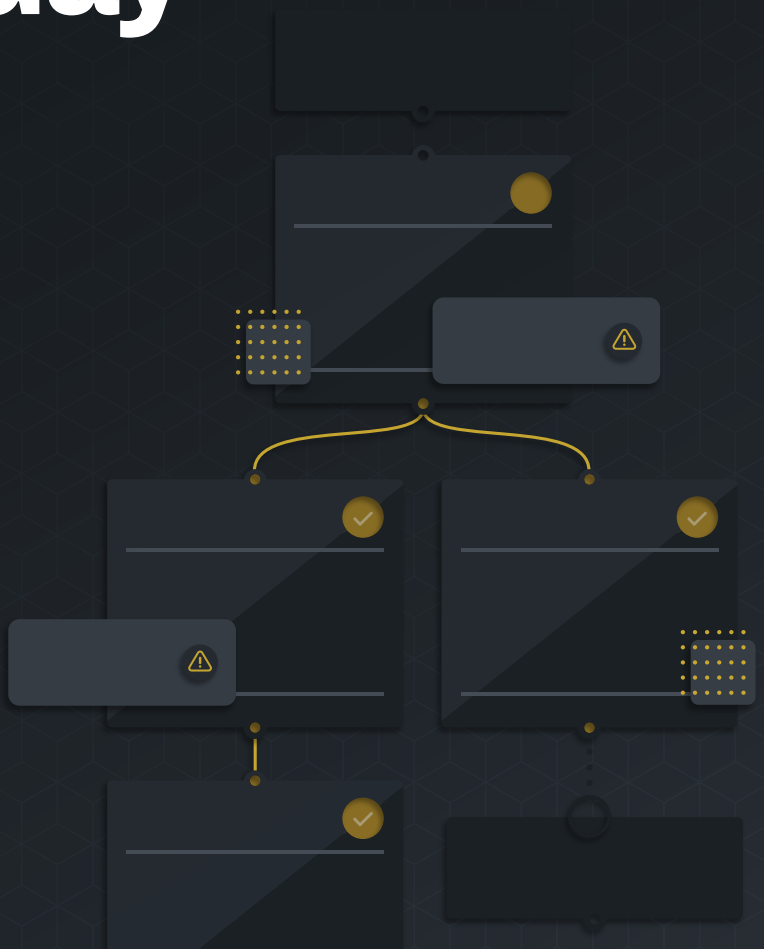
Relatively newer approaches to application security—including security in the DevOps lifecycle (sometimes referred to as DevSecOps) and shifting security left—have suggested significant improvements to the above approaches, but drove little change since the tools and processes themselves remained stagnant.

Part three: Developer-first application security with GitHub

To actually drive down the number of vulnerabilities in production code, security teams need to partner with developers in their preferred environment and leverage their existing workflows. Putting developers front and center for application security is the most effective way to shift security left and succeed against the mounting technical debt that can overwhelm even the best teams.

Using GitHub, your teams can create secure applications with a developer-first approach, empowering your developers to share lessons learned and easily tackle today's application security issues. Instead of relying on multiple tools that cause friction, GitHub offers a unified, native, and automated solution already in your developer workflow, and additional security code reviews during every step of the development process. Developers get security feedback within the development workflow with supply chain and code security features—including code scanning, Dependabot alerts for vulnerable dependencies and Dependabot security updates, secret scanning, and more. You can address security risks earlier to automate vulnerability fixes and ship more secure applications, faster.

Part one: State of application security today



Application security leverages a system of tools, processes, and best practices to manage application-related business risk. Depending on the level of risk you're willing to accept and how critical your applications are, application security ranges from solely being aware of the risks to having well-established processes for quickly identifying and remediating vulnerabilities, ideally before they make it into a production environment.

Modern software is built on open source. Ninety-nine percent of enterprise codebases contain open source code according to Synopsys' 2020 Open Source Security and Risk Analysis Report.[1] But as the adoption of open source components increases, so can security risks for both your developers and security teams due to increased exposure. For example, projects frequently inherit vulnerabilities from unpatched open source components used as dependencies. And the likelihood of these risks is rising, with the 2019 State of the Software Supply Chain Report by Sonatype reporting a "71 percent increase in confirmed or suspected open source-related breaches in the last five years."[2]

Before we dive into different approaches to application security, let's review some common application security concepts:

Static application security testing (SAST)

SAST uses application source code or binary code as input, and scans this code for known vulnerable code patterns to generate results that identify potential vulnerabilities. SAST tools are commonly used in early to late stages of software development, especially prior to shipping the code to production.

1: 2020 Open Source Security and Risk Analysis Report, Synopsys
2: 2019 State of the Software Supply Chain Report, Sonatype

SAST tools run multiple analyzers to find potential vulnerabilities across the code, but the inability to validate context and exploitability may lead to "noisy" results. Since scan results are based on known vulnerability patterns, these results are not highly accurate, with many SAST tools generating false positives. Not only are scans time-intensive, taking anywhere from hours to weeks, but reviewing raw scan results is a labor-intensive task. Your security team or development leads need to validate and prioritize true positives while removing false positives. This ends up becoming the bottleneck for traditional SAST tools.

Dynamic application security testing (DAST)

DAST examines a target application's code to identify its attack surface, or application tree, and deploys the application in a test environment to run simulated attacks. DAST tools are commonly used during QA prior to shipping the code, as well as on production applications.

The process generates raw scan results which point out potentially exploitable vulnerabilities, such as those made available via the user interface. As a result, DAST tools identify a subset of the application layer vulnerabilities reported by a SAST tool, which are known to be exploitable. DAST tools can also find vulnerabilities SAST tools miss, like those related to the running environment of the application (server, frameworks, network). This is why SAST and DAST are used as complementary methods to comprehensively understand the risk posture of applications. DAST tools validate attack results with server responses they receive, so scan results need to be manually reviewed before fixes are planned.

Integrated application security testing (IAST)

IAST finds security vulnerabilities by installing an agent which runs alongside the target application. IAST is commonly used during continuous integration (CI) and quality assurance (QA) phases.

There are two variants of IAST:

Passive IAST is used for applications running in testing environments. When the application goes through use case-based QA tests, the agent identifies potential security vulnerabilities. This approach finds a subset of vulnerabilities that can also be found using SAST or DAST.

Active IAST is used for applications running in live environments and acts as an enhancement for DAST tools. The agent is installed on the running application and performs DAST tests against the application. The agent can view stack trace information and can do detailed behavior analysis on the server side, so the DAST process and results can be improved. Active IAST helps reduce the scanning time and validate attack results for DAST.

Runtime application security protection (RASP)

RASP involves installing an active agent on a running application and using this agent to protect the application at runtime. In contrast to other AST tools, RASP tools are used against active vulnerability exploits on applications running in production environments. RASP agents can detect and prevent predefined

sets of vulnerabilities, but these agents may degrade application performance, especially under heavy usage, DoS, or DDoS attacks.

Fuzzing

Fuzzing (or fuzz testing) uses automated or manual methods to provide invalid, unexpected, or random data as inputs to running applications in a test environment. As these inputs are sent, the target application is continuously monitored for exceptions which may include crashes, abnormal behavior, or potential memory leaks. Fuzzing can provide additional information about a target application and serves as a complementary method for DAST.

Software composition analysis (SCA)

SCA analyzes an application to determine its third-party components, frequently focused on open source software (OSS) security issues and license compliance. SCA is often used in early phases of software development.

Today's SCA tools create an inventory of third-party components and check these components for known vulnerabilities or other operational risks such as license compliance. In some cases, they also offer a library of verified and compliant components for developers to use.

Penetration testing

Penetration testing involves automated and manual tests that aim to test the security controls of running applications. In most cases

penetration tests only cover applications running in production, but they can also be scoped to cover pre-production environments.

Penetration tests can be conducted by internal or external teams, and are typically summarized in reports. The results of these tests are already validated by the testing team, but penetration tests require planning and take longer than automated scanning methods. In addition to technical vulnerabilities, penetration tests can discover faults in the logical flow or user experience of the applications in scope.

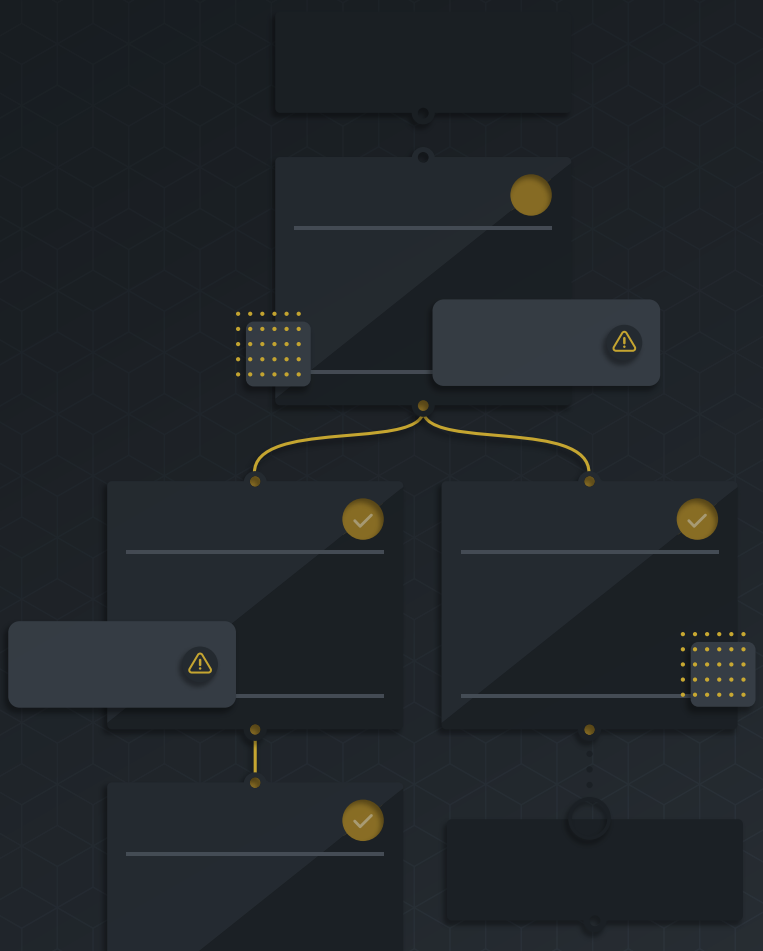
Bug bounties

Bug bounties are crowd-sourced security testing programs which leverage individual security researchers who get paid based on the vulnerabilities that they discover. Bug bounties serve as a complementary solution to all of the methods noted above, but don't typically provide comprehensive coverage for the security posture of applications.

For the average organization today, application security consists of a small set of testing tools integrated with the software development cycle. Depending on your organization's maturity level, tools, and capabilities, application security may either be treated as the final gate before deploying an application, or alternatively as a series of integrated tests as part of the development cycle.

Let's take a look at these two approaches and what they mean for your developers.

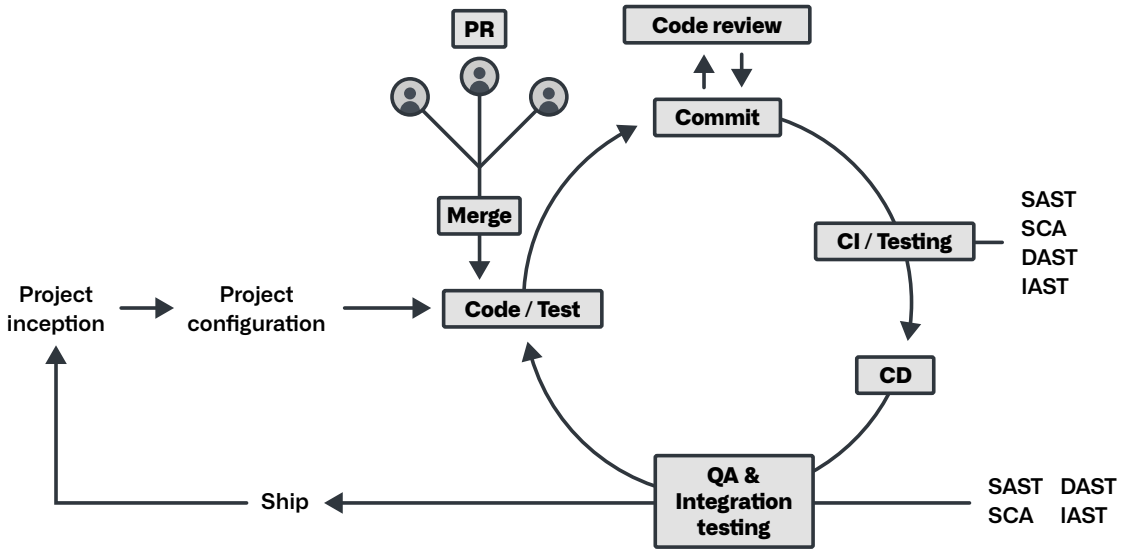
Part two: Traditional vs. end-to-end security



The traditional approach

Having security as a gate prior to deployment is a common approach, and often the first step for organizations just starting with application security. This "traditional" approach consists of a single security test or a series of security tests that take place during the quality assurance phase. These security tests are run by security teams or third parties, and the outcomes of the security results are delivered in bulk to developers for remediation. The tests' findings are then expected to be fixed before the application goes into production.

Application security as a gate



In this traditional gate approach, teams most commonly use SAST, DAST, IAST, and SCA tools.

Although having security as a gate is better than having no application security at all, it causes developer friction and delays delivering secure applications. This is due to three main reasons:

1. **Late security feedback causes confusion**. Security feedback that comes at a later stage in development (often weeks after the code's creation) means that developers have already moved on to the following sprint or the next project, so the vulnerable code in question is no longer top of mind. It can take a while for developers to refamiliarize with the code and context, and the fixes often require additional sprint planning, potentially delaying current projects. "More than 70 percent of all flaws remain one month after discovery and nearly 55 percent remain three months after discovery" per the State of Software Security Report Volume 10 by Veracode.[3]

2. **Scan results have a high noise-to-signal ratio.** Traditional application security tools generate multiple false positives for every true positive, so reviewing scan results is a challenging task. These reviews are generally done by security teams who have limited knowledge about the scanned projects, which makes auditing scan results difficult and labor-intensive. Another approach is to push raw scan results to developers without reviews, but this puts the burden of evaluation onto developers, deprioritizing their effort to actually fix the issue.

   In both cases, a considerable amount of false positives make their way to developers as items to be fixed, causing confusion and frustration.

3: State of Software Security Report Volume 10, Veracode

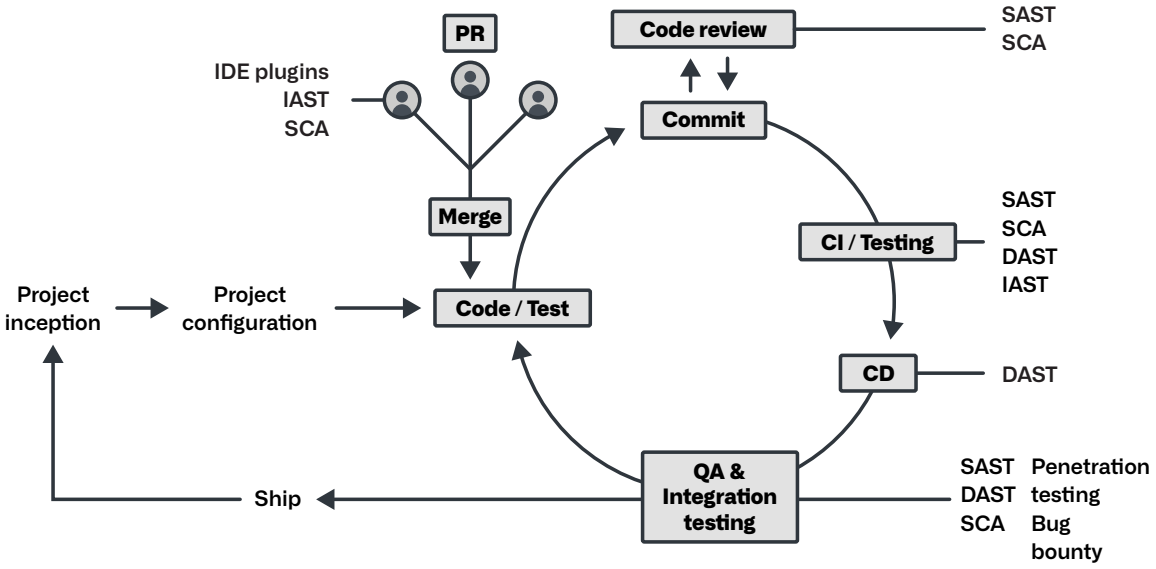
3. **Manual reviews cause bottlenecks.** When scan results are generated by automated tools, they still require manual review to identify true positives and eliminate false positives. Security teams are greatly outnumbered by developers and they simply can't keep up with the sheer volume of raw scan results that need their attention. Manual reviews can take days and even weeks for the average project, creating a bottleneck and delays in project timelines. Delays can be even more frustrating in cases where manual review results don't meet expectations. In most cases, delayed security results mean teams have to ship releases with known vulnerabilities, with no time to fix these issues to meet project timelines.

   Another downside of traditional security is that if a project isn't a priority, it won't get manual reviews. "Raw" scan results may also be shared directly with developers, which are non-validated scan results with a high false positive (non-finding) ratio. Since these results can quickly become frustrating and developers can become desensitized to so many false positives, raw scans are either turned off or their results get ignored. Either option adds more risk to your organization. If security issues aren't addressed in time, they can become a legal liability—like if a security issue is found to be a source of a data breach, or if not remediating a known issue is a breach of one (or more) of your customer contracts.

The end-to-end approach

Organizations that are more mature in application security employ an end-to-end approach, which starts in earlier stages of development and has more points of interaction throughout the development lifecycle.

End-to-end approach



The end-to-end approach provides earlier security feedback to developers and leverages integration and automation capabilities of security tools. But like traditional security, end-to-end security still has several friction points:

1. **Integrations require a lot of upkeep and frequently break because of version updates.** Although security tools are integrated with the development process, these integrations break often. Developers have to drop their current tasks to address a security issue and log into another portal to deal with a different tool or system. Security feedback lacks context and context-switching remains a challenge. The user experience becomes problematic and inefficient for developers.

2. **Security teams and development teams continue to work in silos.** Changing your tooling isn't enough to change

your processes. Silos between the security and application development teams still have to be addressed for end-to-end security to work. Integrating tools streamlines part of the application security process, but this approach again falls short of nurturing more collaboration between security and development teams.

Security teams also still act as reviewers for testing results. For example, every commit on the main branch gets scanned and new alerts are sent to the security team for review. The security team still has to triage and send issues back to developers to fix, and the team likely still has a gating process on release. This is a better approach than having security tests as a gate since some things get caught early, but these teams still lack common processes and platforms to collaborate. With silos and poor communication, issues are pushed back and forth between teams, often leading to delays and sometimes conflict.

3. **Automating traditional tools doesn't solve the false positive problem.** It's exciting to think about automated application security tools. But in reality, automating scans and pushing results to an issue tracker leads to a flood of non-actionable issues. Here the problem is false positives and developers becoming desensitized to noise. With too many alerts, developers ignore test results (and mark them all as "false positives" or "won't fix"). Traditional security tools lack the customizability to adjust sensitivity or improve results over time, so when results are pushed into developer flows, developers switch these tools off.

4. **Traditional tools fail to keep pace with the software ecosystem.** Today's software ecosystem consists of open source, new programming languages, new frameworks, and emerging tools that evolve at a breakneck pace. Since traditional commercial tools are created, updated, and supported by small vendor teams,

they struggle to keep up. There's also very limited communication between these vendors and the developer community, requiring research teams to proactively look for OSS vulnerabilities—an unscalable task with 100 million-plus public repositories. As a result, most commercial tools do well in a few aspects of application security or limit their support to a small part of the software ecosystem. This means your developers and security teams either have to work with multiple tools and vendors for a single use case, or invest in home-grown solutions to fill the gaps that your commercial tools create.

DevSecOps and shifting left

Relatively newer approaches to application security—including DevSecOps and shifting security left—have suggested significant improvements to both traditional and end-to-end security. However, they've driven little change since the tools and processes remain mostly the same.

Even with DevSecOps, traditional and end-to-end security approaches still share common problems:

1. High friction between developers and security teams,

2. Applications frequently shipped with known vulnerabilities (83 percent of applications have one security flaw on initial scan and two out of three applications fail to pass tests based on OWASP Top 10 and SANS 25[3]),

3. Low fix rates for discovered vulnerabilities (only 56 percent of

---

3: State of Software Security Report Volume 10, Veracode

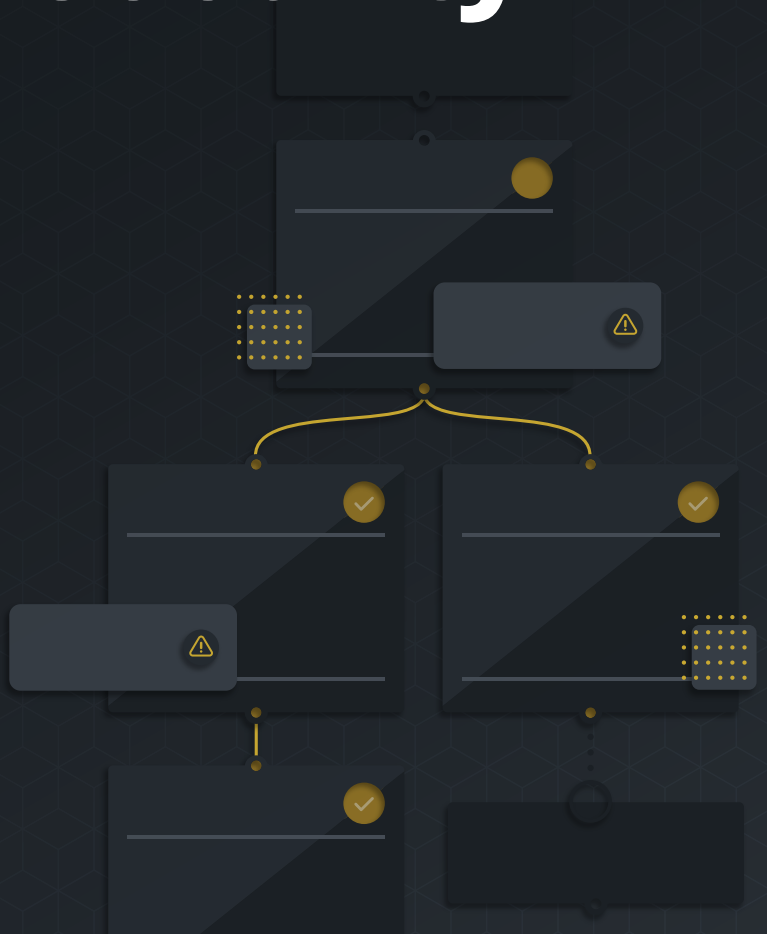
security flaws are fixed, with 24 percent of high-severity flaws left unfixed by developers[3]), and

4. Long exposure periods for detected issues (the median time to fix security flaws is 59 days[3]).

With all of these challenges combined, it's little surprise that web applications have been reported as the main cause of security breaches over the last five years.[4]

3: State of Software Security Report Volume 10, Veracode
4: 2016, 2017, 2018, 2019 and 2020 Data Breach Investigation Reports, Verizon

Part three: Developer-first application security



Developer-first, community-powered security with GitHub

Unless security issues can be identified and fixed by your developers early in the development lifecycle, technical debt will continue to be a challenge for your software ecosystem. And like many other challenges, application security problems are easiest and most cost-effective to solve at the source. To actually drive down the number of vulnerabilities in production code, we need to partner with developers in their preferred environment and use their existing workflows.

> There's only one way to shift security left and succeed against overwhelming technical debt: Putting developers front and center for application security.

By prioritizing developers and giving them the tools to work in the most efficient way, GitHub takes responsibility for making the software we all rely on more secure. GitHub creates secure applications with a community-powered approach that addresses feedback early and often from developers; empowers researchers to enhance and embellish search capabilities; and crowd-sources bug bounty testing programs. Instead of relying on multiple tools that cause friction in the process, GitHub offers a unified, native, and automated solution within the developer workflow. You can address security risks earlier, automate vulnerability fixes, and have better security governance to build and protect applications.

How to improve your project's security with GitHub

1. Start with security in mind

Reviewing security requirements and identifying potential risks with your project prior to implementation are critical steps for preventing expensive vulnerability fixes. Here are a few ways to put them into practice:

Apply security best practices for project configuration

Configuring your project to match security best practices can prevent a lot of problems. Reviewing accounts and access settings (including roles and responsibilities, two-factor authentication, git over SSH, managing teams, integrations, and projects) along with setting a SECURITY.md vulnerability disclosure and reporting policy can go a long way.

Model threats for the project

Software threat modeling is the set of activities that helps identify the potential threats, threat actors, and vulnerable components in a project. Threat modeling requires analyzing the business logic and flow of sensitive data through library APIs like source and sink. Source is the part of code where data is ingested, and sink is the part of code where data flow is completed.

CodeQL is the industry-leading semantic code analysis engine that lets you query code as though it were data—making it easy to discover a bad pattern and then find similar occurrences across your entire codebase. Behind the scenes, CodeQL adaptive threat

modeling semi-automatically boosts your JavaScript security queries with machine learning to find more security vulnerabilities and improve taint specifications. Taint specifications capture the role of library APIs—source and sink—and are a critical component of any taint analyzer that aims to detect security violations based on information flow. A boosted query then produces a ranked list of additional results for you to review. With adaptive threat modeling, your JavaScript and TypeScript queries will identify more security problems. For example, using this technique GitHub Security Lab was able to find 118 new NoSQL injection vulnerabilities across 50 JavaScript projects.

Customize static scans

Custom queries provide a powerful way for static application security testing solutions to detect security issues that may not be covered by the standard rule and/or query sets. These issues can be specific to your codebase or to patterns that are considered issues within context. CodeQL lets you add custom rules easily with a SQL-like query language to focus on issues that matter for your project. You can also configure the CodeQL engine to run the standard query set and custom queries for static scans for the lifetime of your project.

2. Secure every step of the development process

Traditional security approaches have shown that if solutions don't empower developers to find and fix issues early, they're likely to fail. We can also tell that scanning projects for vulnerabilities periodically doesn't stop technical debt from mounting up. As a result, technical debt continues to grow and cause security defects, along with other problems for your team.

GitHub delivers additional security code reviews for every step of the development process with native solutions. Developers get security feedback within the development workflow with supply chain and code security features—including Dependabot alerts for vulnerable dependencies, secret scanning, and more.

Secure the supply chain

A software supply chain is anything that goes into—or affects—your codebase, from development to your CI/CD pipeline to production. Within your supply chain, software dependencies are everywhere. It's normal for your projects to use open source dependencies that you didn't write yourself. The 2019 State of the Software Supply Chain Report by Sonatype reports that anywhere from 85 to 97 percent of enterprise codebases use open source.[2]

If any of your dependencies has a vulnerability, chances are your application has a vulnerability as well. Being able to leverage the work of thousands of open source developers means that thousands of strangers effectively could have control over your production code. Both an innocent mistake or malicious attack to your supply chain can have a widespread impact on your codebase—making security both proactive and reactive. Securing your software supply chain is an ongoing process of knowing what's in your environment, managing your dependencies, and monitoring your supply chain.

Dependency graph, dependency insights, and Dependabot

With features like the dependency graph, dependency insights,

---

2: 2019 State of the Software Supply Chain Report, Sonatype

Dependabot alerts and Dependabot security updates, developers can easily see which dependencies they use and open pull requests with fixes to resolve them automatically. Once you're aware of your dependencies, Dependabot alerts notify you of repositories affected by a newly discovered vulnerability. To do this, GitHub compares the information in the dependency graph to the information in the GitHub Advisory Database. A Dependabot alert can be sent when you've added a new dependency (we check for vulnerabilities in that dependency), or when a new vulnerability is discovered (we alert any repositories that are vulnerable). The alert is sent to repository owners by default.

Dependabot security updates will send you a pull request to update a dependency to the minimum version that resolves a known vulnerability—that is, the first version with the patch. This suggested change to the lock file happens automatically based on alerts.

Secure code

Custom code delivers application logic and unique capability for projects. It's developed by the developers or vendors on your project team. Securing custom code within projects is another critical step in shipping secure applications and preventing potential zero day vulnerabilities.

Code scanning

Code scanning is a developer-first SAST product that's built into GitHub. After it's configured, it scans every code change in your repository for security vulnerabilities and flags them in the developer workflow. This makes it easy to find security vulnerabilities in your code before they ever reach production.

With code scanning enabled, every `git push` is scanned for new potential security vulnerabilities, and results are displayed directly in your pull request. By default, code scanning uses CodeQL, which has an unmatched record finding real vulnerabilities. It includes 2,000-plus CodeQL queries written and open sourced by the GitHub Security Lab and leading security researchers to find potential vulnerabilities in your code with minimal configuration.

Code scanning can also be augmented and expanded to incorporate other commercial and open source scanning technologies like Anchore Container Scan, RuboCop Linting, ShiftLeft Scan, Open Source Static Analysis Runner (OSSAR)—which allows running multiple open source security static analysis tools—and others.

Secret scanning

GitHub has provided secret scanning for public repositories, including API keys and authentication tokens, since 2018. Secret scanning protects our partners and community from unauthorized use of the services protected by those secrets.

As part of GitHub Advanced Security, we're bringing the same lightning-fast scanning engine and broad set of 27 partners (a growing list including all the major cloud providers and many common SaaS providers) to private repositories, so you can catch secrets as soon as they're checked in. Repository administrators will be notified about any commit that contains a secret, and can quickly view all detected secrets in the repository's **Security** tab.

Third-party security capabilities through GitHub Actions

GitHub Actions makes it possible to automate, customize, and

execute your software development workflows in the same place you code. You can discover, create, and share actions to perform any job you'd like, including CI/CD, and combine actions into a completely customized workflow. Since individual actions are reusable as code, it's easy to take advantage of the collective knowledge of millions of other developers and security teams, just as you do in your applications.

Actions serves as an open platform to easily integrate with third-party security tools. Some of the current integrations include OWASP ZAP, which supports both baseline and full DAST scans, SonarCloud Scan, Snyk, and many others.

3. Improve collaboration and get continuous security feedback

Making security an integral part of the developer workflow on GitHub opens up secure collaboration for all: within individual teams, across your organization, and within the community.

Collaborate with security during all stages of development

Getting security input at the pull request level allows developers to have early feedback with the right context at the right time, so they can fix security issues while they're still working on the same portion of the code. Having this enriched view of the security issues also empowers security teams to become a part of the triage and fix process by providing their input on GitHub where and when code is created. When your development and security teams are aligned, they can prioritize issues more efficiently, discuss optimal fixes, and validate results together.

Collaborate between developers, security teams, and the community

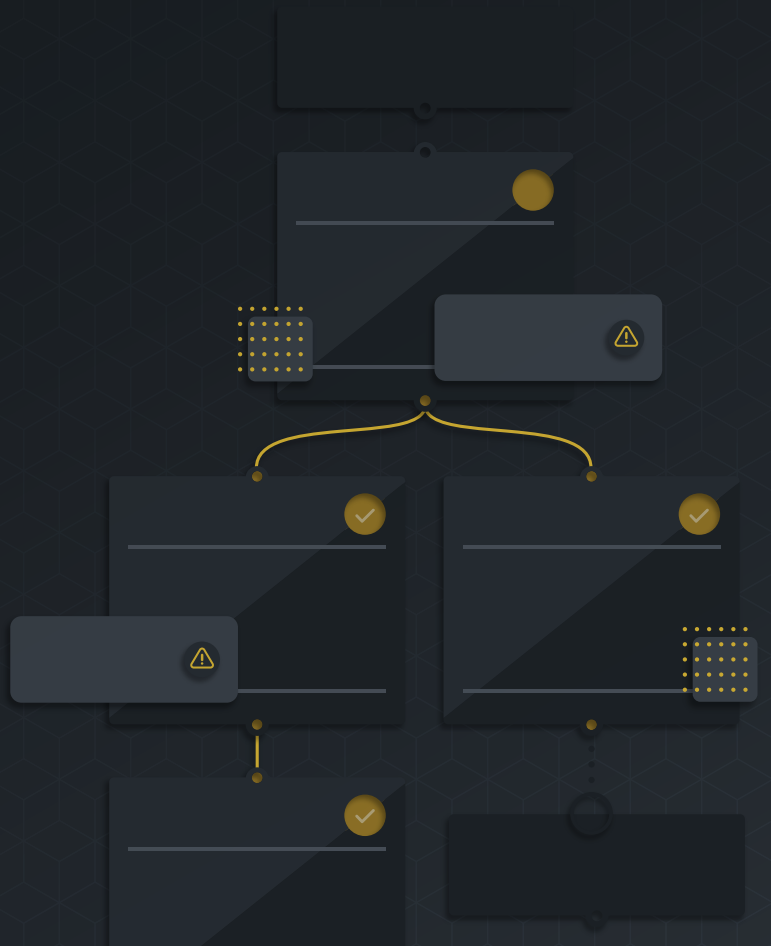
Increased collaboration between development and security teams—plus the customizable nature of CodeQL—helps teams make the most of existing CodeQL queries and create new queries to address their projects' needs. Development teams can identify specific needs for queries, while security champions within development teams or security teams can customize existing CodeQL queries or create new ones. Once you create customized queries, you can then share them across your organization. Your teams can also opt to contribute the queries they've made to the open sourced query set for the community to use.

Alerting the community about security vulnerabilities in your projects is another way security teams, security researchers, developers and the community can collaborate. You can publish security advisories natively on GitHub to quickly notify everyone using the project, and initiate the process to fix these issues. We'll review each published security advisory, add it to the Advisory Database, and may use the security advisory to send Dependabot alerts to affected repositories.

Collaborate within the community: A community- driven approach

The standard CodeQL libraries and queries that power GitHub code scanning are open source and available for anyone to review and contribute—just open a pull request. Since GitHub's security capabilities are open source, when you contribute or review a query, you're also helping secure the software we all rely on.

Conclusion



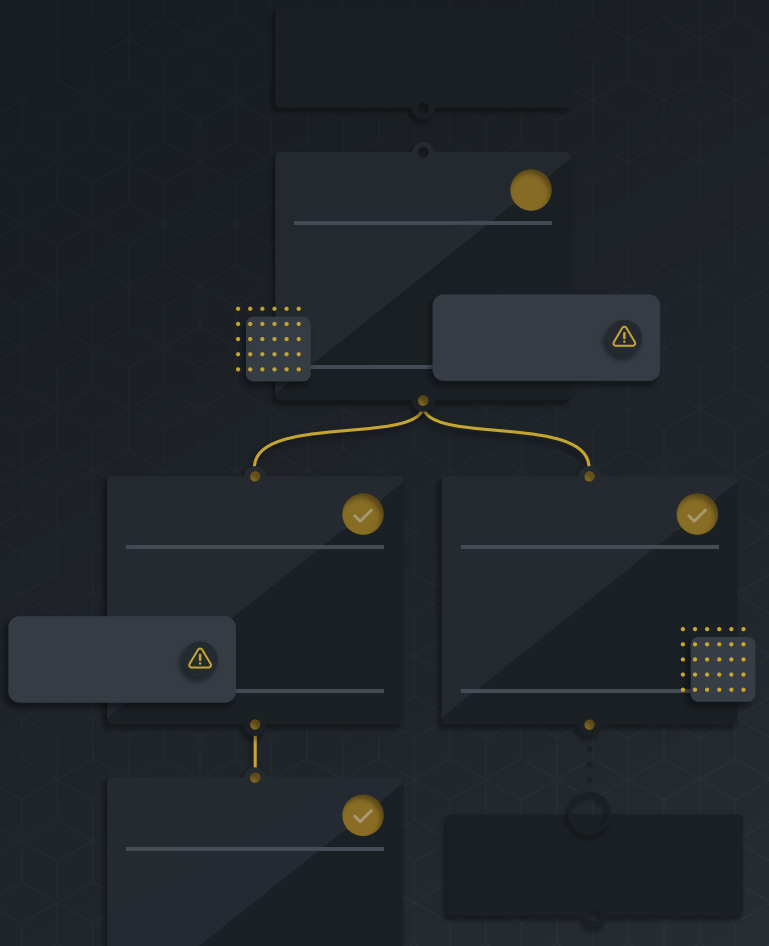
It's no exaggeration: Open source software has changed the world. Many of today's most innovative applications are built on code anyone can freely access and contribute to. Just like open source teams collaborate on shared projects, the only way to combat technical debt with today's increasing code volume and velocity is to solve security issues together. Community-powered security can help security experts share lessons learned and provide better ways to solve today's application security issues.

Developer-focused, community-centric security also makes it possible to find and fix issues earlier, while improving collaboration with both your own organization and the greater open source community. Whether you want to stay up to date with the larger software ecosystem or be aware of the latest threats, GitHub allows you to contribute to the development of security best practices that benefit and empower everyone.

GitHub

Questions about application security?

Learn more at **github.com/learn/security**
or contact our **Sales Team**



WRITTEN BY GITHUB WITH ♥