A Developer's Guide to Application Security

GitLab

What's inside?

Application Security today

Web app and software vulnerabilities are two of the top pathways along which businesses suffer external attacks (Forrester, 2019). Traditional security practices aren't able to scale at the rate that IT shops are expected to deliver new software and updates, largely because testing happens too late in the development process: In fact, 49% of developers encounter the most project delays during the testing stage of the SDLC, according to GitLab's 2019 Global Developer Report. Too often, vulnerabilities are discovered by the security team only after code has been merged and is in a test environment. This forces additional handoff between development and security as teams try to find the defect and make fixes – when developers have often moved on to their next project.

To help quicken the software's journey through security testing, developers are encouraged to write secure code, but many organizations fail to define exactly what that means, and often fail to provide tools to help the developer find and fix security flaws.

What is DevSecOps and why is it important?

Many organizations have adopted DevOps as a means to shorten delivery times and encourage more collaborative and iterative work. The next step is integrating security into DevOps practices, allowing teams to move fast, deliver quality software, and help their businesses avoid catastrophe.

Shifting left

Bringing security practices forward within the development lifecycle is known as a shift left. Developers are beginning to share responsibility for certain security processes, like testing, secure coding, and dependency scanning, and shifting left can make those tasks easier and more efficient.

Development and security leaders should establish policies that use automation to test for adherence during the coding process. Exception-based intervention can be used to tailor policies to different projects on an as-needed basis. Pre-established security policies will also help developers understand exactly what is expected of them when it comes to secure coding. During the development process, automated testing can be applied at code commit, giving developers immediate feedback on the quality and security of their code.

Writing secure code

A lot goes into secure coding. Here are a few best practices to follow:

Security by design

The topic of security should be brought into the very first meeting for every project. Rather than being bolted on at the end, security considerations should be made at every step of the process so that both the SDLC and software itself is, quite literally, secure by design.

Security needs to become a mindset: It's everyone's responsibility, and everyone can contribute.

Security and development teams should have a mutual understanding of each others' requirements, expectations for the project, and intended business outcomes, which can be achieved by general and project-specific policies. It's important that teams establish methods for maintaining secure code through the entire lifespan of the software, so that future updates don't expose critical vulnerabilities. It's also helpful for project teams to understand compliance requirements before coding begins so that software doesn't need to be retrofitted with measures like region-specific data management or user access capabilities.

Minimizing complexity

Simple code is clean code – but simple is a relative term. By minimizing complexity during the coding process, developers can save themselves time in the future when going through remediation. Dependency scanning will help developers identify vulnerabilities within the dependencies they're using, enabling them to fix them early, apply a patch while coding, and then test it as they proceed instead of coming back to it later.

Complexity can also be minimized through the process itself: An integrated toolchain (or a single tool for the entire development lifecycle) will ease the burden of increased security responsibilities and provide a single source of truth to strengthen transparency and improve collaboration between development and security. Manual integration and switching between platforms leave too much room for human error. Auto-remediation will also come in handy by simplifying code feedback and fix mechanisms.

Managing third-party code

Development teams should take an inventory of all of their current third-party relationships. The goal is to gain a thorough understanding of what's being used, who is in charge of the relationship (if applicable), and what data is being accessed and stored.

Regardless of where a breach occurs, if it's your business's data, it's your responsibility.

Moving forward, work with the security team to create formal requirements and standards for all new third parties: Ensure that third-party applications and code adhere to the business' security and compliance requirements. It's also crucial to set data encryption policies (both for internal-only data, and data that get shared with your partners) to protect data based on trigger actions or level of sensitivity.

Testing

Early-stage testing is a great method for shifting left. An integrated, end-to-end tool will make testing especially easy to adopt by enabling developers to code and test within the same interface. When it comes to security testing, there are a number of principles every developer should know:

1. Test early, test often. The sooner developers receive feedback, the better. Building upon code that has already been reviewed and verified will help teams cover more code and save both time and resources down the road.

Mature DevOps practices are 90% more likely to test 91-100% of all code than early-stage DevOps practices.

Source: GitLab 2019 Global Developer Report: DevSecOps

2. Breadth before depth. Before deep-diving into any of your code, mitigate risk by testing every code change for the most common vulnerabilities (such as the OWASP Top 10) first.

3. Keep an automated log of every code deployment, dependency, and update. By providing a record of every certified change and requiring approvals for resolving critical vulnerabilities, a master log will both improve transparency and reduce risk. This will also help simplify the audit process should inquiry become necessary.

4. Diversify your security scanning. A single type of test or scan isn't going to reduce much risk – defense in depth requires multiple scans, such as those outlined in the table below. Employ a diverse set of tests to cover your bases.

Security tests and scans for the developer's toolkit	
SAST	Static application security testing (SAST) looks for vulnerabilities in the code itself.
DAST	Dynamic application security testing (DAST) looks for vulnerabilities in how the code functions.
Dependency scanning	Dependency scans look for vulnerabilities in third-party code.
Container scanning	Container scans look for vulnerabilities in the container image and registry.
License compliance	License compliance tools look for compliance around the use of third-party code.

Compliance

A complete compliance strategy includes both regulatory and self-imposed policies. Standards set by governments or governing bodies often cover the rules you need to live by, but aren't much help when it comes to quality and user experience. It's important to make compliance requirements well-known before coding begins so that they can be built into the workflow and product without impeding user experience.

Zero Trust for applications

Zero Trust is exactly what it sounds like – a framework for trusting nothing. This includes segmented infrastructure, data encryption and classification, and access levels approved only on an as-needed basis. Zero Trust is also a good mechanism for establishing a security mindset and culture: It encourages team members to approach projects with the expectation that what they're working on will, at some point, be attacked by malicious actors or exposed under insider threats.

Zero Trust can and should be used to assess potential vendors, partners, and third-party functionality on whether they either fill a need in protecting your product, or will fit into your security framework. Some important application control mechanisms include auditing, emergency change control, identity and access management (including access control mechanisms like multi-factor authentication), data inventorying and flow mapping, data loss prevention, and data archiving. With cloud native applications, there is also a strong need for instrumentation and monitoring the app's infrastructure – your containers, orchestrators, and cloud services.

Deploying secure apps

Serverless computing, cloud-native, containers, and Kubernetes are changing how apps are deployed and managed, and have also expanded attack surface and complexity for every business. Cloud providers and orchestrators have some out-of-the-box security capabilities, but they won't cover everything. It's up to your project team to check for and fill any open gaps. New tools and services also offer significant opportunity for misconfiguration (like the recent CapitalOne breach) that could leave both employees' and customers' personal information at risk of exposure.

Take protective measures

To counter these risks, your team can take the following actions:

- **Stay current with system updates and patches.** New vulnerabilities are constantly found in open source software, bringing the risk of zero day attacks. It's therefore important to update and patch as soon as possible.
- **Bring security into each software iteration through automated testing.** Secure coding doesn't stop at launch – updates, however small, should follow the same security testing and standards as the initial app development process.
- **Secure horizontally before narrowing in.** Mission critical apps are certainly important, but stopping your security efforts there will leave massive opportunity for malicious intent. Instead, start by testing for common vulnerabilities across as many apps as possible, and then dig deeper where you need to.
- **Simplify your workflow.** Stay in the same tool as much as possible to make the process easier: A streamlined toolchain (or single tool) will boost the likelihood of thorough testing, and give a more complete picture of security vulnerabilities – made actionable for the developer.

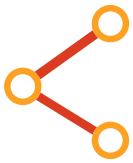
Securing the DevOps lifecycle with GitLab

Developers can best achieve secure practices when using a tool that comprises everything they need to get the job done. That's where GitLab comes in. With every code commit, developers receive results from automated tests, review the results themselves, scan dependencies and containers, and collaborate directly with their security peers. Everything we do will help you deliver quality product faster and with less risk.

Want to learn more?

Attend GitLab's premier user conference in San Francisco, January 14

Read more on DevSecOps and GitLab's own security journey



Connect with us on Twitter and LinkedIn

Try GitLab Gold for free


GitLab