GitLab

Guide to Software Supply Chain Security

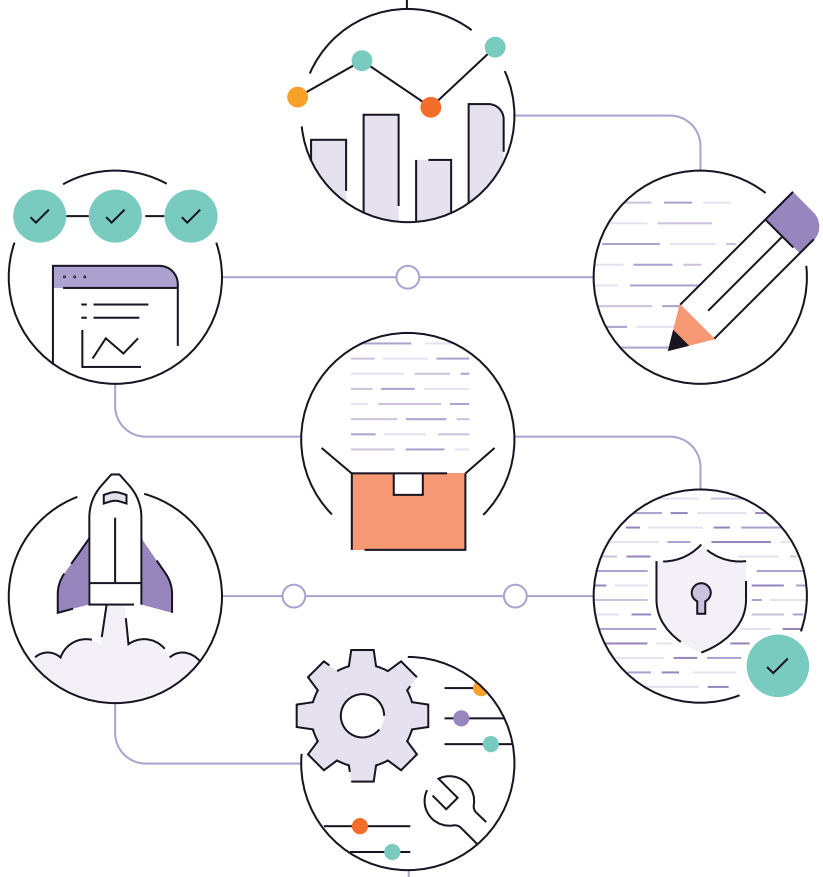


Table of contents

Introduction

Securing the software supply chain is too often an afterthought. However, high-profile attacks, such as those carried out on SolarWinds and the Colonial Pipeline, are proving too costly to allow security to be kicked down the road in the software development process. The U.S. government is now demanding via an executive order that organizations become better stewards of the software supply chain. Are you ready?

This guide will help you understand the imperative to develop strong protection strategies early in the software development cycle, some of the security (and DevOps) terminology you need to know, and the tools that can help identify vulnerabilities in the software supply chain to mitigate risk. We also have included a quiz to help you assess the security of your software supply chain.

Software supply chain security

Software supply chain security is a system of checks and balances that kicks off in the initial stages of software development. Before your first line of code is even written, you should be thinking about the following precautionary steps:

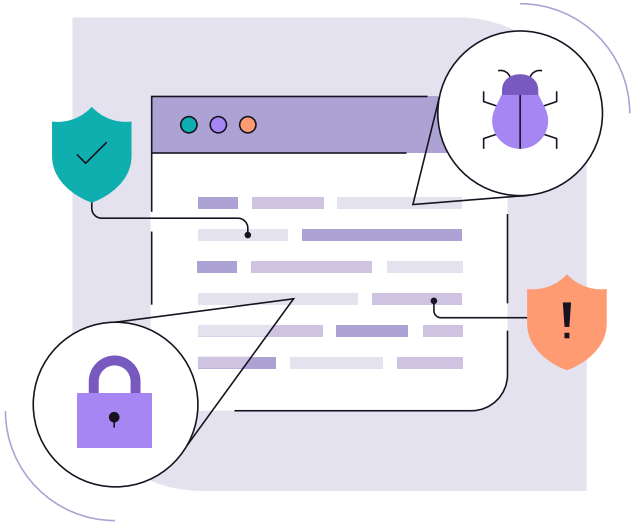
- Who will have access to the code (internal and external parties)
- Who will have ownership over code approvals
- How you will create a chain of custody and version control
- The basic security steps necessary to ensure malicious code can't be injected into your product
- Mechanisms for responding if code is somehow altered by bad actors

Failing to take these steps can lead to attacks, such as ransomware demands, that impact your organization or, worse yet, your ecosystem of partners and customers.

You can protect the software supply chain by implementing DevSecOps security-as-code best practices, which include the use of automation and compliance controls, visibility into your inventory of code, integration of multiple types of security scans into the build, test, and deploy environments, and extension of security to your container and cloud environments. We will dig into all these later in this guide, but first, consider why software supply chain security is front and center today.

The SolarWinds attack

In 2020, attackers inserted malicious code into SolarWinds' Orion Platform that allowed them to gain entry into the internal environments of organizations running Orion software. The highly sophisticated attack, known as SUNBURST, is now seen as an inflection point for software development practices that reinforced the need for all organizations to consider the security of their software supply chain.



The colonial pipeline attack

In 2021, the Colonial Pipeline, a large fuel supplier to the East Coast of the U.S., was attacked by hackers. Although only its billing and accounting systems were attacked, not the operational systems, the company shut down production and delivery for multiple days and paid a $4.4 million ransom (in BitCoin) to prevent the release of 100 gigabytes of data. The ransom was later recovered by the FBI but the incident revealed how vulnerable the nation's vital infrastructure is to attacks and how quickly a supply chain can be impacted.

There are many more examples of hackers being able to exploit lax security practices in software development to wreak havoc on companies and their supply chain. No business wants to be in the position of explaining to customers how their lack of security policies and procedures throughout the development lifecycle led to a critical attack.

U.S. government action on software supply chain security

In May 2021, Pres. Joseph R. Biden, Jr., signed the Executive Order on Improving the Nation's Cybersecurity, which includes a section on enhancing supply chain software security. The executive order and the initiatives it sparked call for guidance on standards, procedures, or criteria regarding:

- Who will have access to the code (internal and external parties)
- Who will have ownership over code approvals
- How you will create a chain of custody and version control
- The basic security steps necessary to ensure malicious code can't be injected into your product
- Mechanisms for responding if code is somehow altered by bad actors

Impact beyond the U.S.

What the U.S. does in terms of applying better security to supply chains will have an impact far beyond federal agencies and software vendors that sell to them, and well past the country's borders. Just as General Data Protection Regulation (GDPR) reached beyond the European Union, U.S. regulatory changes will have a similar reach back across the pond.

Speaking the same language

DevOps and Security teams must be able to communicate with one another to begin to address the challenges of securing the software supply chain. Below is some common terminology that will help.

5 security terms every DevOps pro should know

1. False positives can occur when vulnerability scanning identifies a potential security flaw. Sometimes false positives are in the eye of the beholder in that a vulnerability may be concerning to one enterprise and totally acceptable to another. This happens because risk acceptance thresholds are not universal. It can also happen when vulnerability scanners lack calibration or necessary acuity.

2. Role-based access controls (RBAC) determine access to applications and resources based upon predefined roles. People are assigned to roles and their role determines what they can do. It's an important part of common controls for compliance with most regulatory requirements. It also works hand-in-hand with zero trust. Careful RBAC management can help protect your software supply chain.

3. Vulnerability management offers developers, project leaders and security teams alike end-to-end situational awareness by constantly monitoring the entire application lifecycle from development through to deployment and production. This visibility is essential to uncover security risks and resolve them according to security and compliance policies. It also helps security teams improve efficiency and reduce risks in the process. It's somewhat analogous to Value Stream Management used by DevOps.

4. Vulnerability scanning is a security method that inspects applications for weaknesses that hackers could use to attack the system. A best practice is for developers to scan their own code changes for flaws — like coding bugs, unpatched vulnerabilities, misconfigurations and unprotected secrets — and then fix them before hackers find and exploit any vulnerabilities. The term "shifting left" means identifying vulnerabilities during development when their remediation is more efficient and new technical debt can be avoided.

5. Zero trust is a network security strategy designed to give only authenticated and authorized users and devices access to applications and data. When applied to modern cloud-native environments, zero trust means protecting access from people and machines (for example, APIs), with the assumption that hackers will infiltrate the network. It requires thoughtful protection against lateral movement and privilege escalations.

5 DevOps terms every security pro should know

1. Breaking changes happen when a change made in a single part of the software system leads other parts of the system to fail, often in a costly cascade that can lead to disruptions and time-consuming rework. Simply put, a breaking change happens when backward compatibility is not maintained. A weighty application development concern, breaking changes often happen in shared code libraries accessed by multiple applications, during security patches, and while changing or deleting parts of an API. To avoid the problem, developers should review changes for quality and security, be diligent with documentation, plan out API changes, and track vulnerabilities.

2. CI pipeline automates the integration of code changes via merge/pull requests that apply quality tests, vulnerability scans, and other common controls such as approvals for policy exceptions. The CI pipeline is sort of the assembly line of the software factory. Best practices will standardize the processes and rules that are applied. Often, security teams will want to break a pipeline when vulnerabilities are found, but development may prefer that security non-critical flaws are just automatically captured for later resolution.

3. Infrastructure as Code (IaC) is a way to manage and store IT infrastructure specifications — networks, virtual machines, application environments, etc. — as an easy-to-copy-and-distribute code file. Without IaC, managing the infrastructure is a manual process, leading to problems with availability, scalability and inconsistencies. A key DevOps practice, IaC minimizes environmental drift by using consistent deployment configurations. It enables DevOps teams to test applications in production-like environments, standardize outcomes, and lower the cost of infrastructure management.

4. Minimum Viable Product (MVP), or Minimal Viable Change (MVC), which is based on the idea of continuous improvement, is an agile development methodology in which products are quickly developed with only a minimal number of must-have features. The results are faster software releases and avoiding time wasted on unnecessary features. MVP also allows developers to quickly get user feedback, as well as insight into how the features are used, which can guide future builds — all with the least amount of time, cost, and effort. Security and automation need to be part of this rapid iterative development.

5. Value Stream Management uses metrics, captured as byproducts of DevOps automation, to improve efficiency, reduce bottlenecks, and get a clearer overview of the development process. Process improvements can be made in a methodical, data-driven manner. Advanced enterprises will incorporate vulnerability scanning and remediation processes within their end-to-end view of the software development lifecycle (SDLC).

Tips for securing the software supply chain

A key aspect of securing anything is the use of good cyber hygiene practices. The following are foundational to security programs of any maturity level:

- Strong passwords
- Timely application of software patches
- Multi-factor authentication
- Key rotations
- Secret detection

The more an organization does upfront to establish — and automate — security standards, the easier it will be to identify anomalies or threats and act swiftly to mitigate them.

In addition to basic security hygiene, we will cover seven tips for improving the security of your software:

1. Apply common controls for security and compliance
2. Automate common controls and CI/CD
3. Apply zero-trust principles
4. Inventory all tools and access, including IaC
5. Consider unconventional scans to find unconventional vulnerabilities
6. Secure containers and orchestrators
7. Generate a software bill of materials (SBOM)

1. Apply common controls for security and compliance

Achieving regulatory compliance and ensuring proper security relies on managing control points throughout your software supply chain, along with the visibility necessary to audit the results. For example, you must set controls for who can make changes to code and configurations, approve merge requests (that may have policy exceptions), and scan applications for vulnerabilities. Some of the common controls you'll want to think about include:

- Segregation of incompatible duties
- Identity and access approval controls
- Configuration management and change control
- Access restrictions for changes to configurations and pipelines
- Protected branches and environments
- Auditing
- Licensed code usage
- Security testing

When it comes time for an audit, you'll need to have a way to see who changed what, where, and when — as well as who reviewed, approved, and merged it — along the entire software development lifecycle. Auditors should be able to check the compliance of every audit event in your logs or you should be able to prove that you have an automated process in place that satisfies the requirements.

2. Automate common controls and CI/CD

Automating policies for common controls helps you ensure more consistent compliance, reduce your audit surface, and more easily prove compliance to auditors. One popular way to do this is by automating CI/CD to apply common controls.

For example, security scanning for vulnerabilities is a common control you can automate. First, determine which projects or applications you want to require scans on and what the policies for those scans will be. Hopefully, your policies will require using more than one type of scan. CI templatves can be used to ensure automated scans are consistently applied. Without automation in place, it's all too easy to end up doing scans less frequently than you planned and let a vulnerability go too long and too far into the software supply chain without being remediated.(See page 9 for more on the types of security scans available.)

3. Apply zero-trust principles

Zero trust is an approach that assumes that hackers are going to get inside your network and focuses on protection from the inside rather than just the perimeter. Embracing this perspective can protect you from lateral attacks where hackers find an easy way in that may be a low-value asset but use privilege escalation and advanced techniques to reach mission-critical apps and data.

The modern software era relies upon so much more than application code and the network, which makes zero-trust principles a necessity. Today's complexities include APIs, secrets, containers, orchestrators, cloud services, templates, and other tools that your development team uses. All of these provide additional attack surfaces. If an attacker exploits misconfigurations of one of these elements of your software supply chain, they may move laterally across applications, clusters, and environments.

Here are some key ways zero-trust principles help you secure your software supply chain:

- Lateral movement becomes more difficult because each service has to be authenticated
- Protection is consistently applied for both human access and machine access (such as APIs)
- Stolen credentials are less valuable.
- Non-targeted attacks are less successful
- Role-based access prevents threats from malicious or simply careless insiders

4. Inventory all tools and access, including IAC

First, you'll want to inventory anything your development team is using that could be a point of entry or new attack surface — from code repositories to APIs, containers, orchestrators, artifacts, templates, and all your build tools. In many organizations, the security team isn't aware of a lot of the tools that are being used, and you can't make something more secure if you don't know it's there.

Once development and security teams are on the same page, it's time to re-examine all your access controls. Certainly, take a look at typical things like your network and endpoints, but keep going until you've covered all points of entry along your software supply chain. For instance, consider who can change IaC templates. Containers are used to stand up new projects quickly. If they are not scanned, security flaws can be quickly replicated across projects. Be sure to scan containers and monitor the behavior of APIs and orchestrators. For some ideas, check out the National Institute of Standards (NIST) Secure Software Development Framework (SSDF).

5. Consider unconvetional scans to find unconventional vulnerarabilities

Conventional scans, like static analysis, dynamic analysis, secret detection, dependency scanning, and container scanning, will help you find Common Vulnerabilities and Exposures (CVEs), or vulnerabilities with a known signature. But vulnerabilities that don't have known signatures or are unique to your environment could also expose the organization to a breach or attack.

The line between what is a security flaw and what is a logic flaw can be thin — and won't matter if the flaw is exploited.

For unknown vulnerabilities, fuzz testing can be especially helpful. There's coverage-guided fuzz testing, API fuzz testing, and protocol fuzz testing, and they allow you to find insecure logic flaws that do not have a signature of a known CVE. To help shift this security check earlier in the software development lifecycle, it's a good idea to add fuzz testing within your CI pipeline right alongside other security scan types. (See page 9 for more on fuzz testing.)

Other ways to find unknown vulnerabilities in your applications include:

- Monitoring for stability and reliability issues, which can lead to exploitable vulnerabilities
- Confirming reproducibility to minimize false positives and identify root cause analysis

The goal is to test wider and deeper so you can maximize your testing coverage and reduce your overall risk.

6. Secure containers and orchestrators

Protecting your software's infrastructure in production — like Docker containers and Kubernetes — is an important piece of securing your software supply chain. To keep things secure and compliant, you'll want to apply many of the same common controls in code creation, testing and deployment to your container infrastructure as well.

Make sure you include container scanning to find any known vulnerabilities in your container images. If you're using Kubernetes, you'll also want to scan your Helm charts, and you can do that with static application security testing (SAST).

Apply zero-trust principles and work to limit lateral movement among your containers. You can create a policy that blocks east-west traffic, so pods will only be allowed to communicate with certain other pods.

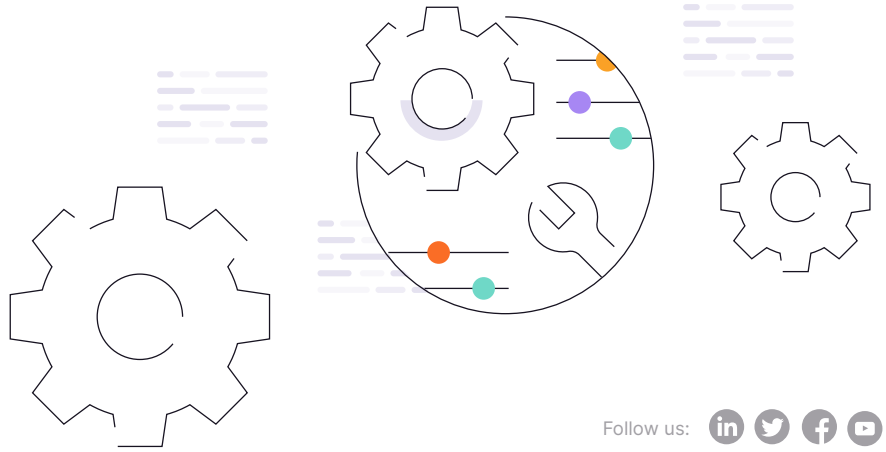
Also, think about more obscure things like the container registry and who has write access in your organization. A compromise of one person could potentially lead to a compromise of the container registry, which could lead (via pipelines) to compromises of numerous projects.

7. Automate SBOM generation

A software bill of materials (SBOM) is a list of all the components in a codebase — essentially a list of the ingredients that make up your software.

By automating SBOM generation, you can avoid lengthy manual processes to confirm that malicious software is not packaged within your software. Automating SBOM generation will provide insights into dependencies across transient structures from package managers and containers. Developers will be able to expedite remediation activities when SBOM vulnerabilities are displayed in the UI.

To further increase usability and adoption, reduce the number of tools used for reviewing and processing SBOMs. Utilizing SBOM in an end-to-end secure platform helps to protect from multiple attacks, including protection for internal code, external sources, and even the build process.



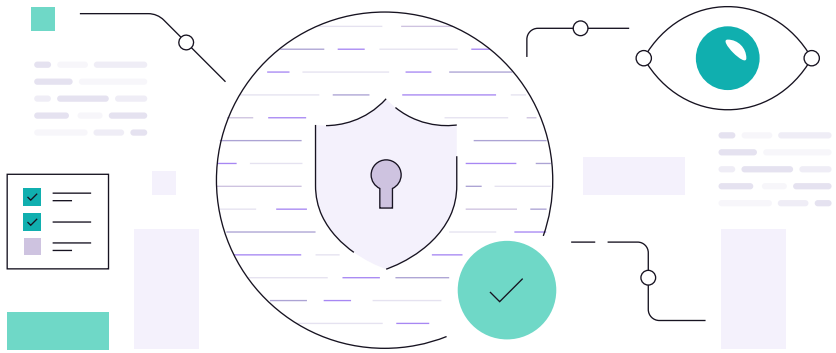
Types of security scans for the software supply chain

Security scans provide a way for teams to get visibility into the health and safety of the software supply chain. When it comes to security scanning, the more scans, the better. Security pros call this Defense in Depth. Each scan type can find different types of vulnerabilities. No one scan type is sufficient on its own.

Here are some common security scans, each of which provides distinct and critical information about the software being changed or surrounding code upon which it relies:

Container scanning

Container images may be based on images with vulnerabilities or simply have vulnerabilities on their own. Either way, it's key to analyze your containers for known security vulnerabilities alongside other security scans automated in your CI pipeline.



DAST

Dynamic application security testing (DAST) analyzes your running web application for known runtime vulnerabilities. DAST happens during the QA stage and can be run within the CI pipeline before a merge when using a review app, or used outside of a pipeline to continuously monitor live applications. DAST can find things like cross-site scripting errors or broken authentication issues.

Dependency scanning

Some experts suggest software dependencies can actually be an application's largest area of vulnerability. So scans that analyze external dependencies (for example, libraries like Ruby gems or Apache) for known vulnerabilities on each code commit are critical. Dependency scans can run while applications are being developed and tested and are an ideal solution for teams using open source libraries. A dependency scanner should look at all vulnerabilities, including nested or transitive dependencies.

Fuzz testing

Fuzz testing offers what no other security scan does: ilt "pings" an application with unexpected or even "malformed" data in an effort to get it to crash, thus measuring the stability of the application. There are two types of fuzz testing: Coverage-guided fuzzing, which looks at the source code while an application is running, and behavioral fuzzing, which tries to bring out the differences between how an application actually works versus how it is expected to work. Fuzz testing is a great choice for teams looking to unearth previously unknown vulnerabilities and logic flaws.

License compliance

An increasing reliance on open source code means teams need to be able to easily track license compliance. When code is committed, project dependencies should be searched for approved licenses as well as any licenses that have been disallowed by an organization's general policies. While less of a security issue, ensuring proper license usage can avoid the risk of non-compliance.

SAST

Static application security testing (SAST) scans the application source code and binaries to check for weaknesses and vulnerabilities. SAST scans are run before code is deployed and can find dangerous attributes in a class or other unsafe code. Ideally, SAST results should be easily and regularly available to developers within their existing workflows (such as in their pipeline) so necessary changes to code can be made as the code is being worked on. Contextual results are vital if teams want developers to take on increased ownership of code safety and can greatly improve the efficiency of remediations.

Secret detection

It can be a simple matter for a developer to unintentionally include sensitive data like passwords, tokens, and other credentials in a remote repository. Checking for these secrets in code commits and in project histories allows teams to proactively resolve these issues before anything sensitive is improperly disclosed.

Platform features that help secure the software supply chain

Securing the software supply chain requires access to certain tools that security teams typically use to identify and prioritize risks as well as create an audit trail. To ensure your software is being properly monitored, make sure your DevOps platform has these features:

Security dashboard

A security dashboard lets you know the number of vulnerabilities that were introduced over time. For instance, after you run a security scan, the detected vulnerabilities should automatically be displayed in the dashboard. Dashboards should be able to be customized so that departments, teams, and individuals can easily see the vulnerabilities in their projects. Vulnerabilities also should be labeled according to their severity levels.

Vulnerability management

Vulnerability management lets you dive deeper into the information that appears on the security dashboard. With so many moving parts, it's critical to have a comprehensive plan to deal with vulnerabilities. Teams need to be able to see the problem areas, triage them for severity/threat status, be able to note trends, and then track the status and remediate or resolve the vulnerabilities. A solid plan and the correct tools will give teams full visibility into an organization's risk and can unite developers and security around a common view.

You should be able to sort and filter vulnerabilities by characteristics such as severity, status, and the type of scan performed. Without moving to another tool, the reporting should show if there has been activity on the vulnerability — for instance, whether an issue has been created to resolve it, a developer has been assigned, or it has a targeted completion date. Also, if it's been dismissed, why and by whom.

Audit logs and audit events

Audit logs and events offer detailed information about what types of changes were made on the system, when, and by whom. For example, audit events should show if a vulnerability check or license check was added, or if a group was added or removed. If a policy stops the build and an exception is approved, audit logs will show who approved the exception. This comprehensive audit trail can greatly simplify an audit, while also expediting root cause analysis following a breach.

Dependency lists

A dependency list shows all the code dependencies throughout the software so that if a vulnerability is detected, you know the extent of its reach and impact. You should be able to drill down and see what file or files the dependency is in as well as what licenses it is associated with.

Test your software supply chain security readiness

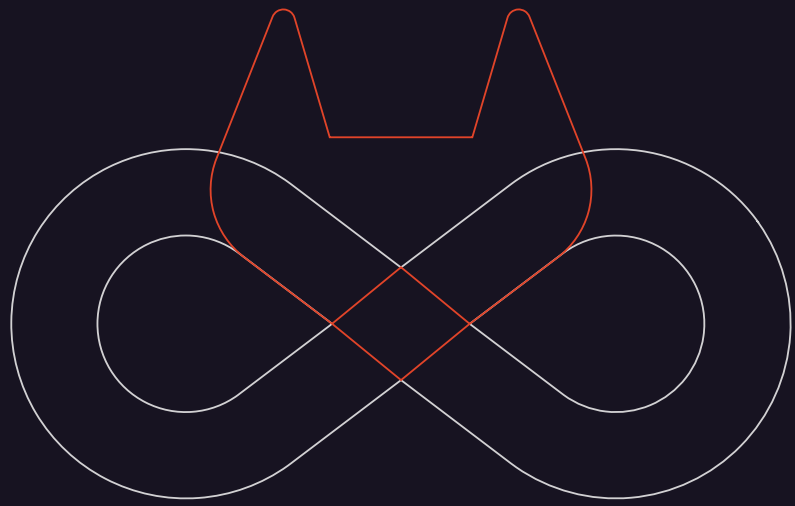
In just nine questions you can see how your team ranks!

Take our two-minute quiz here and be directed to security content designed for what you're experiencing today.

About GitLab

GitLab is The One DevOps platform for software innovation. As The One DevOps Platform, GitLab provides one interface, one data store, one permissions model, one value stream, one set of reports, one spot to secure your code, one location to deploy to any cloud, and one place for everyone to contribute. The platform is the only true cloud-agnostic end-to-end DevOps platform that brings together all DevOps capabilities in one place.

With GitLab, organizations can create, deliver, and manage code quickly and continuously to translate business vision into reality. GitLab empowers customers and users to innovate faster, scale more easily, and serve and retain customers more effectively. Built on Open Source, GitLab works alongside its growing community, which is composed of thousands of developers and millions of users, to continuously deliver new DevOps innovations.



GitLab


GitLab