A Field Guide to Threat Vectors in the Software Supply Chain

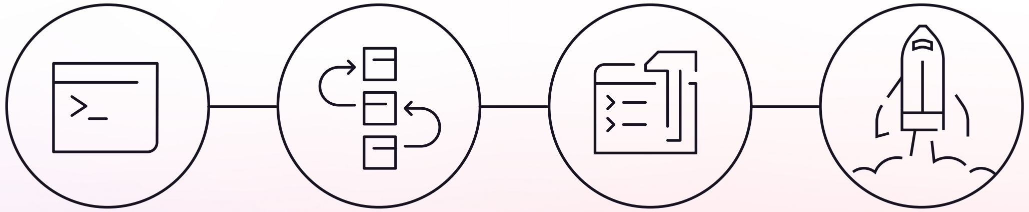


Table of contents

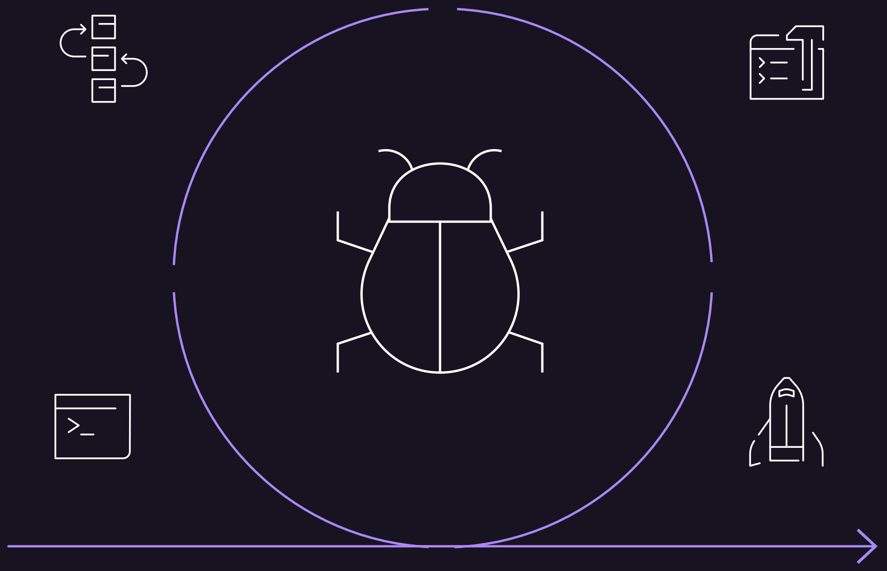
Introduction

Software isn't developed in a vacuum. An entire ecosystem of components — the **software supply chain** — is involved in building, testing, and delivering a piece of software. This ecosystem provides fertile ground for developing a new application, with abundant open source packages, libraries, tools, and processes. But there are dangers, too. The software supply chain is a complicated web of relationships and dependencies that can be exploited by attackers, and recent high-profile attacks are forcing organizations to take a serious look at how they can become better stewards of the software supply chain.

Threats can enter the software supply chain at four main points: through vulnerabilities in the **source** code of the software itself; through vulnerabilities in **dependencies** such as open source components that the software uses; through vulnerabilities in the software **build** pipeline; and through insecure configurations in the application after **release**. These four steps (**source, dependencies, build,** and **release**) are like a long, winding river, and attacks at any stage can impact everything downstream — including customers, their customers, and end users.

The key to ensuring the security of each step in the software supply chain is to enforce zero trust principles: securing access to the source code by authenticating, authorizing, and continuously validating all human and machine identities operating in an environment; verifying that no open source or other dependencies used in a piece of software contain known vulnerabilities; preventing unauthorized access to build pipelines; and testing configurations and APIs for weaknesses. Essentially, scrutinize everything and everyone — human, machine, open source component, or application configuration — for potential threats.

This guide will help DevOps teams understand the types of threat vectors that can emerge at each stage of the software supply chain and steps to take to mitigate that risk by establishing zero trust. As you explore each page, think about whether your organization is equipped to identify and remediate each type of threat — compromised source control, risky open source dependencies, compromised build pipeline, and insecure web applications — and consider ways to incorporate software supply chain security into your software development lifecycle.



Compromised source control

Identity and access management (IAM) represents the biggest attack vector in the software supply chain, according to the **Cloud Native Computing Foundation**. The source code is the foundation of the supply chain, and it is essential to ensure the source code's security and integrity by closely managing who has access to the code and how changes to the code are reviewed and merged. If attackers gain access to source code management (SCM) systems, they can modify the source code and take over repositories, impersonate users, and modify downstream aspects of the software build process, such as the CI/CD pipeline.

Not all compromised code is deliberate. Mistakes happen. A well-meaning developer may write custom code that contains unintentional vulnerabilities, such as flaws in logic or poor encryption. Or, someone might inadvertently commit secret information such as SSH keys or access tokens to the source code repository without encryption. But no matter the intent, these "**zero day**" — or as-yet undetected — vulnerabilities can create opportunities for attackers to gain access to the software supply chain.

Kaseya VSA ransomware attack (2021)

Flaws in authentication logic are like unlocked doors that can give attackers access to the entire software supply chain. In July 2021, the REvil ransomware group was able to abuse an authentication bypass vulnerability in **Kaseya VSA (Virtual System Administrator)**, a remote monitoring and management software package. The vulnerability allowed the attackers to gain an authenticated session and upload a malicious payload targeting managed service providers (MSPs) and their downstream customers. In all, it's estimated that the attack affected **50 direct Kaseya customers and between 800 and 1,500 businesses further down the software supply chain**.

Tactics

- Implement a **version control system** that **considers security and compliance throughout the software development lifecycle**. The system should include access controls such as **multi-factor authentication (MFA)** and **expiration of access tokens** once a task is complete.

- Require **signed commits** by configuring push rules to reject commits that are not cryptographically signed. A **cryptographic signature** provides extra assurance that a commit originated from an authorized user rather than an intruder.

- Set up **review and approval rules in merge requests** to **enforce additional approval** for requests that would introduce security vulnerabilities or software license compliance violations.

- Automate **security scanning and testing** such as **Static Application Security Testing (SAST)** and **Dynamic Application Security Testing (DAST)** to identify bugs and potential vulnerabilities before software release, and perform **secret detection scanning** to avoid committing credentials and other sensitive information to the source code.

Risky open source dependencies

Software consumers have high expectations, and to keep up, organizations need to be able to build and release software quickly. By building on top of open source components, software developers can save time and effort — in fact, open source is increasingly being seen as **a requirement for innovation**. The problem is, it can be difficult to know for certain what's hiding in that code. Just as failing to manage the quality of goods used in a manufacturing process will jeopardize the quality of the final product, using open source code without validating the quality and security of that code can open the door to cyber attacks.

Dependencies in open source components can be **direct dependencies**, meaning open source packages that are included directly in a project, or **indirect dependencies**, which are dependencies of dependencies. An open source dependency is "risky" when it presents a threat to downstream components in the software supply chain, including end users. Risky dependencies can either be unintentional flaws that are found and exploited by attackers, or malicious code deliberately inserted by attackers into public libraries and open source projects to gain access to downstream targets.

Most modern cloud-native applications are developed using a multitude of open source components, creating a tangle of direct and indirect dependencies. Managing this complex environment can be a daunting task. To ensure the security of the software supply chain, it's important to verify that all open source dependencies contain no disclosed vulnerabilities, come from a trusted source, and have not been tampered with.

Log4Shell vulnerability (2021)

Zero-day vulnerabilities in popular open source projects can be massive attack vectors. In November 2021, the Alibaba Cloud security team reported a new vulnerability in the extremely popular open source **Apache Log4j logging framework**. Later dubbed Log4Shell by security researchers, the vulnerability allowed attackers to **steal passwords and sensitive data and infect networks with malicious software**. Because Log4j is such a popular open source tool, used across many suppliers' software and services, Log4Shell was quickly recognized as a major threat to the software supply chain. In the months after the vulnerability was discovered, **Microsoft observed rampant Log4Shell exploit attempts**, including attacks by sophisticated adversaries such as nation-state actors.

Tactics

- Run **software composition analysis (SCA)**, including dependency scanning, against any dependencies within a project. **Dependency scanning** identifies project dependencies and checks whether there are any known vulnerabilities in those components that may affect the main application. Manage the results of these scans and require approval for merge requests when new vulnerabilities are identified.

- Generate a **software bill of materials (SBOM)**, which is essentially a **list of software ingredients used in a project**, including application and system dependencies. The SBOM makes it easy to review all the packages included in the project along with their version and any detected vulnerabilities.

Compromised build pipeline

The build pipeline is the assembly line of the software supply chain: where all the software components are assembled into a deployable package. If the build pipeline is compromised, attackers can inject malicious code into the build process and thereby distribute that code to downstream components of the software, including end users. Because CI/CD pipelines have network access to a range of environments and credentials to pull dependencies and push artifacts, these pipelines are one of the most dangerous attack surfaces in the software supply chain.

The build pipeline comprises a number of components, each of which needs to be secured. **Build steps** represent specific tasks performed at each stage of the pipeline. **Build workers**, also called **runners**, are applications that carry out each task. **Build tools** are programs that automate the process of building and ensuring the integrity of the final software artifact. And the **pipeline orchestrator** is the tool that deploys build steps and workers and manages the overall CI/CD workflow. Vulnerabilities in any of these components can give attackers control over the software built by the pipeline or access to secrets used in the pipeline.

SolarWinds hack (2020)

One of the biggest cybersecurity breaches of the 21st century began with a compromised build system. Attackers first gained access to the build system belonging to U.S. technology firm SolarWinds in late 2019. In February 2020, the attackers injected malicious code known as SUNBURST into Orion, SolarWinds' network monitoring software, that allowed them to modify software updates SolarWinds was pushing out to Orion users. The attack went undetected for months, allowing attackers to spy on SolarWinds customers — including private companies and the U.S. Government — as well as gain access to the data and networks of their customers' customers and partners.

Tactics

- Adopt a systematic approach to ensure **security of self-managed runners**. Avoid running containers in privileged mode, and run Docker containers with the `--privileged` flag enabled only on isolated and ephemeral virtual machines. Be careful when cloning runners to ensure that multiple runners don't end up with the same authentication token.
- Generate **release evidence** — a single artifact produced for each software release that documents the chain of custody for assets, commits, and issues. Include the results of CI tests and security scans and publish the release evidence on a shared platform to create a single source of truth that can be furnished during an audit or in the compliance process.
- Verify that **build artifacts** have not been tampered with by adhering to Supply-chain Levels for Software Artifacts (SLSA) and SigStore standards — security frameworks that help ensure the security and integrity of the software supply chain. When evaluating a CI/CD solution, choose a platform that can support the generation of attestation or provenance by integrating the CI pipeline with a tool capable of signing the build output.

Insecure web applications

Even if the source code, dependencies, and build pipeline are free from vulnerabilities, attackers can still wreak havoc by exploiting weaknesses in an application's design or security configurations. Imagine an ATM skimmer: a malicious device, disguised to look like part of an ATM, that collects sensitive information such as card numbers and PINs. The skimmer doesn't modify the inner workings of the ATM itself, but it still manages to steal sensitive information by hiding in plain sight. In the same way, attackers can steal private data, gain unauthorized access to accounts, or impersonate legitimate users, all without touching the source code or software build pipeline.

Injection attacks such as **cross-site scripting (XSS) attacks** allow attackers to manipulate an application so that it returns malicious scripts to users. XSS and other injection attacks represent a massive potential attack surface — injection ranks third on the Open Web Application Security Project (OWASP) Top 10 Web Application Security Risks. Other web application security risks on OWASP's Top 10 include **security misconfigurations** (insecure configuration options within an application, such as disabled or improperly configured security features, that can be exploited by attackers) and **authentication failure** (flaws in identity, authentication, and session management that allow attackers to impersonate legitimate users).

Mirai botnet (2016)

Insecure network devices represent a potential army just waiting for an attacker's command. The Mirai malware, first identified in 2016, turns Internet of Things (IoT) devices with insecure default settings into remotely controlled bots that then launch a large-scale network attack — often a distributed denial-of-service (DDoS) attack like the one that took down DNS provider Dyn in October 2016, leaving much of the internet inaccessible on the east coast of the U.S. Although the original creators of Mirai were caught and sentenced in 2018, the botnet lives on, with new variants emerging each year that continue to pose risks to vulnerable network devices.

Tactics

- Use a **secure CI/CD tunnel** to automatically deploy policies for open source cloud-native networking, observability, and security tools to ensure least access network firewalling and to detect and prevent intrusions.
- Use **operational container scanning** and **DAST** to detect security vulnerabilities in running applications, including misconfigurations of the application server or incorrect assumptions about security controls that may not be visible from the source code.
- Use **web API fuzz testing** — essentially, feeding random inputs to the application to see how it behaves — as part of the CI/CD workflow to discover bugs and potential security issues that other QA processes may miss.

Final thoughts

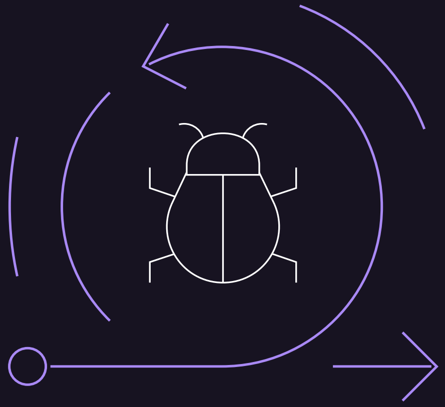
Taking the steps to recognize threat vectors in the software supply chain ensures that the software development lifecycle remains an engine of innovation and drives benefits for the business, rather than being a potential backdoor for attackers. Start identifying the threats in your software supply chain by reflecting back on zero trust at every stage in the supply chain, and adopt the tactics shared in this guide to minimize risk and mitigate vulnerabilities that might already exist.

As a next step, consider **adopting a DevOps platform to protect against software supply chain attacks**. Combining a powerful DevOps platform with a holistic security program can help you quickly gain control and visibility of your software supply chain by:

- Helping to shore up security hygiene
- Automating scanning, policies, and compliance
- Protecting application infrastructure
- Securing the software factory itself
- Iterating with continuous assessment and improvement

Additional resources

- **Test your software supply chain security know-how:** Understanding the software supply chain and how to secure it is imperative for DevOps teams today. Take this quiz to see how prepared you are for the challenge.
- **The GitLab 2022 Global DevSecOps Survey**: In May 2022, over 5,000 DevOps professionals shared details about their teams and practices. They told us that secure software development is now the number one reason for – and benefit of – DevOps platform usage.
- **DevSecOps with GitLab**: Integrating security into your DevOps life-cycle is easy with GitLab. Security and compliance are built in, out of the box, giving you the visibility and control necessary to protect the integrity of your software.
- **The ultimate guide to software supply chain security**: Pairing DevSecOps with software supply chain security makes processes repeatable, increasing security and reducing the opportunity for human error or malicious activity. This comprehensive guide dives into all aspects of software supply chain security.



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