WHITE PAPER

Software Supply Chain Security

Table of contents

Executive brief

Software supply chain attacks are not a new security concern, but recent high-profile attacks such as SolarWinds, CodeCov, and Kaseya have brought the topic to the forefront of cybersecurity awareness across the globe.

> ***Gartner predicts that by 2025, 45% of organizations worldwide will have experienced attacks on their software supply chains, a three-fold increase from 2021***

Supply chain attacks have not only increased in volume and frequency, but have also become more sophisticated, involving Advanced Persistent Threat (APT) actors. This trend, together with the potentially wide impact of a singular successful supply chain attack, requires organizations to take dedicated steps to ensure the security and integrity of both the software they build and supply to their customers, and/or the software they procure for internal usage.

This paper seeks to provide organizations with a guide to this perplexing topic. It introduces the concept of the modern software supply chain, characterizes attacks targeting the software supply chain, and outlines the risk these attacks pose. As of the beginning of 2022, standardized protection and mitigation workflows are still being formulated as part of various federal and community initiatives, and measures such as producing and maintaining an SBoM (Software Bill of Materials) and security automation have emerged as best practices. This paper provides an overview of these principles to help organizations design and implement a supply chain security framework.

Introduction

Software supply chain attacks might feel like a new phenomenon but in truth they have been around for quite some time. By their very nature, these attacks target the components, processes, and people involved in building software, so they can be dated back to the advent of software.

Still, one cannot deny that they are becoming more and more frequent, and although the number of known, successful supply chain attacks remains relatively small, the impact of these attacks has been extensive.

In December 2020, FireEye, a cybersecurity company, discovered a backdoor in SolarWinds' Orion software. Subsequent research revealed that the backdoor was inserted by the attackers by accessing the software's build server. Using the software's routine update mechanism, the backdoor spread to the software's various users. It granted the attackers access to networks, systems and data of their victims.

One of the largest and most consequential software supply chain attacks to date, SolarWinds affected a long list of government agencies and corporations, including the U.S. Pentagon, Department of State, Department of Homeland Security and Microsoft.

While SolarWinds is one of the largest successful supply chain attacks to date and represents a watershed moment in the history of cybersecurity, it is not a singular event and must be seen as part of a growing list of software supply chain attacks.

The European Union Agency for Cybersecurity (ENISA), for example, identified 24 software supply chain attacks taking place between 2020 to July 2021:

SUPPLIER	SUPPLIER CATEGORY	YEAR	IMPACT	ATTRIBUTED GROUPS
Mimecast	Security Software	2021	Global	APT29
SITA	Aviation	2021	Global	APT41
Ledger	Blockchain	2021	Global	-
Verkada	Physical security	2021	Global	Hacktivist Group
BigNox NoxPlayer	Software	2021	Regional	-
Stock Investment Messenger	Financial Software	2021	Regional	Thallium APT
ClickStudios	Security Software	2021	Regional	-
Apple Xcode	Development Software	2021	Global	-
Myanmar Presidential Website	Public Administration	2021	Regional	Mustang Panda APT
Ukraine SEI EB	Public Administration	2021	Regional	-
Codecov	Enterprise Software	2021	Global	-
Fujitsu ProjectWEB	Cloud Collaboration	2021	Regional	-
Kaseya	IT management	2021	Global	REvil Group
MonPass	Certificate Authority	2021	Regional	Winnti APT Group
SYNNEX	Technology Distributor	2021	Regional	APT 29
Microsoft Windows HCP	Software	2021	Global	-
SolarWinds	Cloud Management	2020	Global	APT29
Accellion	Security Software	2020	Global	UNC2546
Wizvera VeraPort	Identity Management	2020	Regional	Lazarus APT
Able Desktop	Enterprise Software	2020	Regional	TA428
Aisino	Financial Software	2020	Regional	-
Vietnam VGCA	Certificate Authority	2020	Regional	TA413, TA428
NetBeans	Development Software	2020	Global	-
Unimax	Telecommunication	2020	Regional	-

Supply chain attacks identified from January 2020 to early July 2021 (ENISA)

This trend has helped highlight the need for a structured response by policymakers and the security community.

Recognizing this need, the U.S government issued an executive order calling for the modernization of the its cybersecurity; better communication and collaboration on cybersecurity between the Federal Government and the private sector. Aiming to protect government agencies from future software supply chain attacks, the order further instructs NIST to establish best practices, guidelines, and criteria for upcoming standards. Software suppliers will need to comply with these standards, or they will no longer be able to sell software to the Federal Government.

Responding to this executive order, representatives from technology providers, financial services, telecom, and cybersecurity companies signed up to support the Linux Foundation's OpenSSF—a cross-industry collaboration to develop improved tooling, training, research, best practices, and vulnerability disclosure, and to tighten the security of open source supply chains, minimizing the risk of attacks.

These initiatives are part of a necessary worldwide, cross-industry effort to formulate standardized tools and methods to protect against software supply chain attacks. But building those frameworks will take time. In the meantime, organizations can devise and implement a software supply chain security approach based on lessons learned from previous attacks as well as existing security tools and methodologies. This addresses current needs and prepares organizations to comply with new standards in the future.

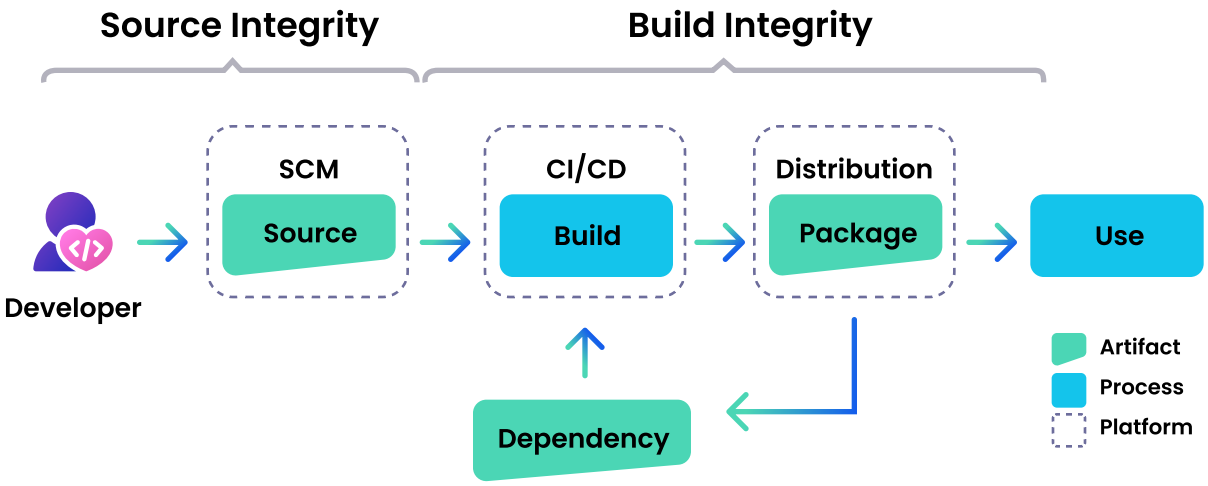
What is the software supply chain?

Put simply, a software supply chain is the sum of all activities required by an organization to build, produce and distribute software.

The easiest way to understand a software supply chain is via comparison to manufacturing supply chains.

In these traditional supply chains, a variety of activities transform raw materials into finished products. For the car you drive, the supply chain includes all the parts used to assemblie it; but it also includes the people, tools, and processes involved in producing the car, taking it to market, and selling it.

In comparison, in the software supply chain, development processes transform code into software. For the application you are using (or building), the supply chain consists of code, binaries, and other components. It also includes the development teams, tools, and processes for building, packaging, and deploying the application, and the infrastructure used to run it.



Modern software development has made the software supply chain more complex than it was only a few years ago. There are multiple reasons for this growing complexity:

Product innovation - consumers today expect intuitive, feature-rich, and cutting-edge products. This puts software vendors under increased pressure to deliver more innovation, quickly and reliably.

External services - to enable fast delivery, organizations are more inclined to outsource elements not core to their business by embedding external services, such as payment, navigation, speech-to-text translation providers, and others.

Technology - technology is evolving at an unprecedented pace. New operating systems, processors, graphic chips are providing new possibilities that were unknown up until only a few years ago. As an example, the mobile phone has become a standard interface for countless products and services. But the average lifetime of a mobile phone is 2.5 years.

Process - the process of building software is now based on modern methodologies and practices like agile development, CI/CD, and DevOps, which have together resulted in an accelerated pace of delivery and time to market.

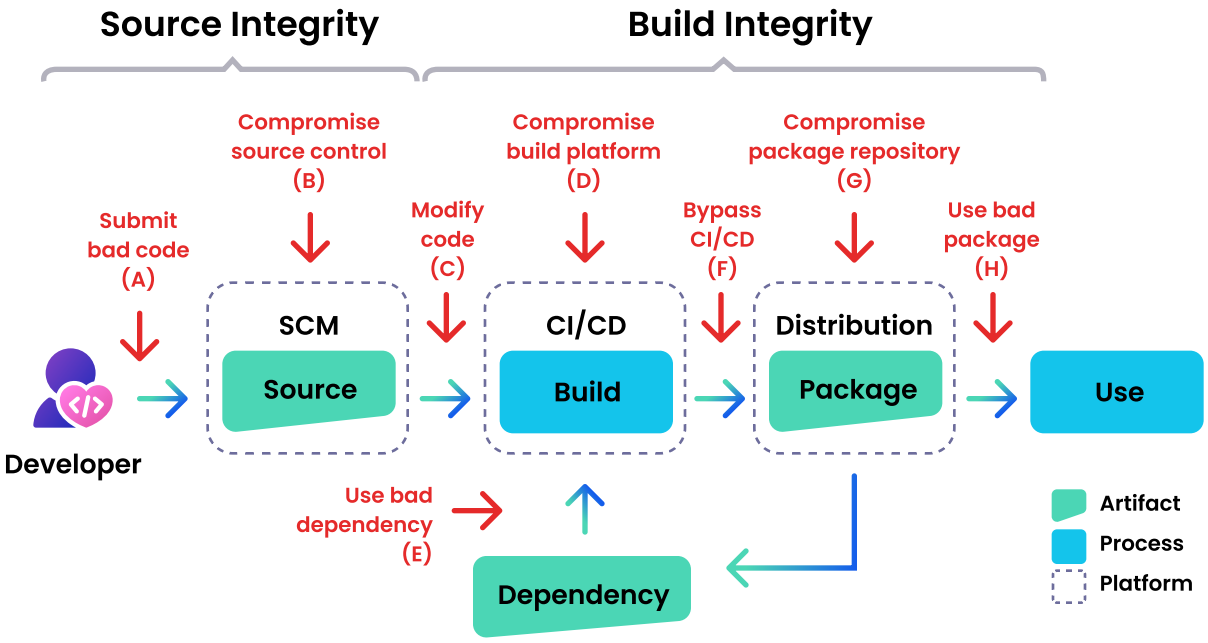
Code - the actual code used to build an application is an assembly of a much longer list of components, including custom code, open source dependencies, build and packaging scripts, containers, and infrastructure provisioning configurations (i.e Infrastructure as Code). The list goes on.

Suppliers - these ingredients originate from a more diverse and distributed list of sources. They could be privately hosted code repositories, but more often than not they come from cloud-based, even public, sources.

These changes and trends have resulted in extremely complex software supply chains and have helped make them an attractive attack vector and target for malicious actors.

Software supply chain attacks

In a software supply chain attack, attackers use malicious code to compromise an “upstream” component in the chain with the end goal of compromising the target of the attack: the “downstream component”. Compromising the upstream component is not the end goal; it merely opens a window of opportunity for the attackers to compromise the target of the attack, by inserting malware or providing a backdoor for future access, for example.



Any one of the links making up the software supply chain can be compromised. Indeed, there are as many possible supply chain targets as there are types of software. But current research highlights three main targets: dependencies, pipelines, and the combination of the two: pipeline dependencies.

Dependencies

In this type of attack, application dependencies such as open source packages or container images pulled in by developers to build software are compromised. Attackers use a variety of different methods to insert malicious code into publicly accessible packages. As a result, malicious code is automatically downloaded by unsuspecting developers using the malicious package. Dependency confusion is one such method, overriding privately-used packages with malicious, public packages using the same name.

Pipelines

In this type of attack, the development pipeline used to build and release software is compromised. Attackers use a variety of methods to inject malicious code into the code that defines the build process, such as CI scripts, build tooling configurations, and infrastructure as code. As a result, the build process itself is compromised and used to distribute malicious code to downstream consumers.

Pipeline dependencies

In this type of attack, the external dependencies used as part of the build pipeline are compromised. These could be, for example, 3rd party plugins, tooling binaries, or the build environment itself. In the Codecov breach, attackers obtained credentials from a Docker image and used them to compromise an ”upload bash script” used by Codecov customers.

Supply chain attack spotlight: `event-stream`

Given the prevalence of open source in modern applications—open source is used in 98% of applications—open source packages are increasingly becoming a popular vector for software supply chain attacks.

Open source packages, especially due to how they are managed and used in applications, have proven to be an efficient conduit for distributing malicious code. Packages are uploaded to registries with little to no security oversight, and are downloaded—millions of times a week in the case of popular projects—into codebases with almost the same amount of scrutiny. For this reason, we are seeing more and more cases where popular package registries such as npm, PyPI and RubyGems are used to distribute malicious packages.

In this section, we will focus on one of the more notable software supply chain attacks that leveraged the popular `event-stream` package.

What is `event-stream`?

The `event-stream` package was a JavaScript toolkit that provided utilities for creating and managing streamsin JavaScript applications.

Why was it targeted?

At the time of the attack, `event-stream` was used by 1,600 packages and downloaded 1.5M times a week, which means the blast radius of the attack was huge. Likewise, the package was not being actively maintained—while there were regular releases of the package up to version 3.3.4, that last release occurred over two years before this incident. This enabled the attacker to gain control of the package.

How was the attack mounted?

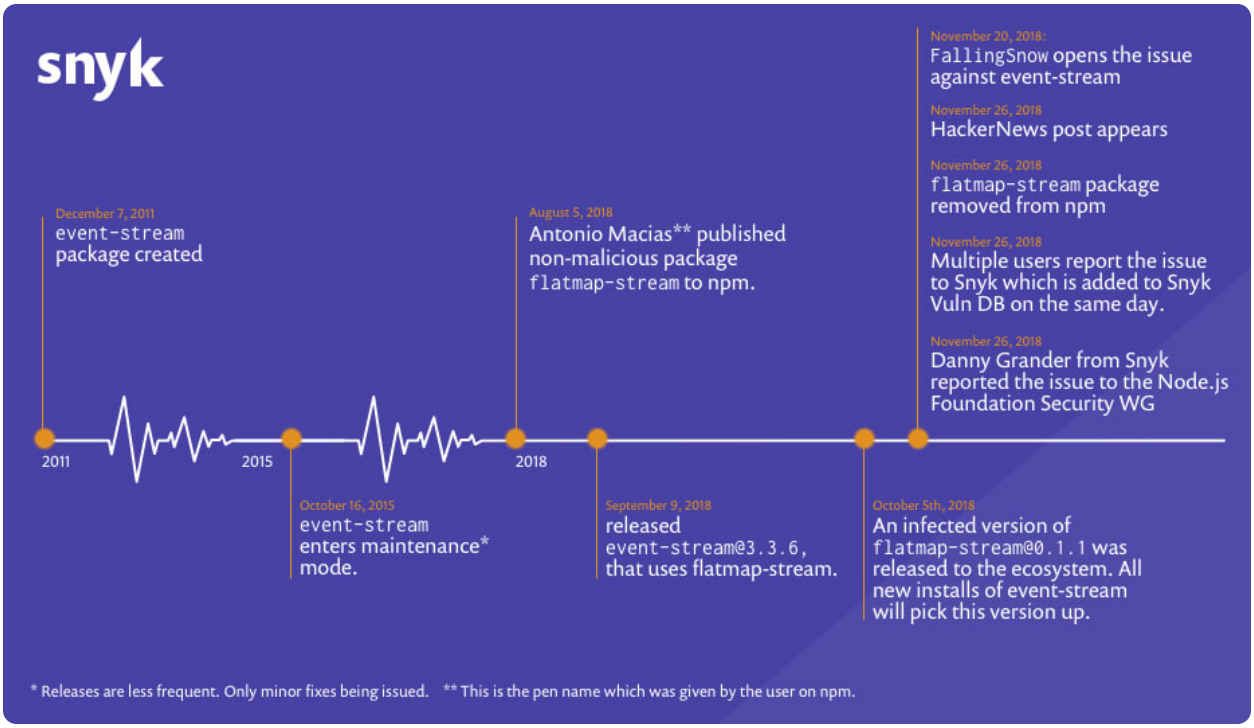
First, using various methods of manipulation and subterfuge, the attacker managed to become a contributor to the package and gained full npm publishing rights. In September 2018 he modified the package to depend on another package called `flatmap-stream`. One month later, this package was altered to include malicious code which immediately affected all subsequent downloads of `event-stream`.

What was the target of the attack?

As already mentioned, the target of such attacks is usually not the original “upstream component” but the “downstream component.” In this case, the attack targeted Copay, a secure bitcoin wallet platform which was successfully infiltrated in versions 5.0.2 and 5.1.0. The full details on the backdoor inserted by the malicious package are available in this blog post.

What was the result of the attack?

At the time, `event-stream` was used by thousands of other JavaScript packages and was downloaded millions of times a week. It took about a month and a half before other users noticed the malicious code. By then, this malicious version of `event-stream`, version 3.3.6, had been downloaded millions of times.



A framework for software supply chain security

While software supply chain security is still very much an active field of research, and there are a variety of models being proposed, this document aims to provide a list of common recommended best practices that make up a framework for software supply chain security:

Secure your code - as the raw material used in the software supply chain, organizations need to apply the correct control to custom code written by developers and the third-party components interwoven into and used by this code.

Secure your pipelines - the different tools and processes making up the modern development workflow are increasingly targeted by attackers and should be mapped out and secured as part of a supply chain security framework.

DevSecOps - based on a tight alignment between security and the overall software development process, the DevSecOps model can help organizations to understand, manage and mitigate supply chain risk. It also enforces guardrails to ensure adherence to the two practices described above without slowing down development.

Securing your code

Code constitutes the raw material used to build the end product and is a key ingredient in the software supply chain. Code can be developed in-house or it can be pulled in from external sources. Whatever the case, ensuring code integrity should be one of the fundamental steps taken to tighten software supply chain security.

Open source

Using an SCA tool, teams can integrate security testing early and across the software development lifecycle to manage and mitigate risks in the open source packages in an application. This includes vulnerabilities in packages pulled into an application indirectly, by transitive dependencies. Snyk research has found that 80% of vulnerabilities in open source packages are introduced by transitive dependencies, further illustrating the risk facing organizations using open source and the need for SCA tools. SCA tools can also detect open source software licenses to help organizations ensure they are compliant with the legal requirements of open source packages.

Key recommendations:

- Use an SCA tool to detect third-party open source packages, including those used indirectly (transitive dependencies).
- Scan in multiple places in the software lifecycle: IDE, SCM, CI/CD.
- Keep open source packages up to date to ensure both the health and security of applications.
- Verify, define and prioritize pulling open source packages from trusted sources.
- Train and enable developers to conduct thorough research into open source packages before usage.

Containers

Containers are another possible weak link in the software supply chain. Security risks can be introduced via your own code, Linux packages, open source libraries inside a container, the methods containers use to interact with the host operating system and adjacent containers, configurations for networking and storage, and more. Besides vulnerabilities introduced directly by the code and tools added to an image, issues can originate from other images that a container relies on (parent or base images).

Base images from trusted providers should be free from malicious software, but still often have vulnerabilities in the Linux packages and developer tools they supply. Containers are freely shared, and an attacker can create a seemingly useful container and share it, but it might harbor bitcoin miners or other malware.

A container security tool can help manage and mitigate security risk in a container image, and ideally should also identify the base image to ensure that it is trusted. The tool should also identify the application components in containers, especially when direct access to the source code is not available.

Key recommendations:

- Start with base images from a provider you trust. Use digital signatures to verify authenticity.
- When possible, opt for minimal base images that have only the basic operating system packages, and your framework version of choice, and then build up from there.
- Check your images for vulnerabilities early and often. Scan in multiple places in the software lifecycle: the desktop, in CI, stored images in registries, and the containers / pods actively running in your clusters.
- Rebuild, test and redeploy containers often, even if your software has not changed. New vulnerabilities will be discovered and if you are using base images from trusted sources, those vulnerabilities will be fixed as the Linux maintainers provide patches.
- Monitor containers over their lifecycle to catch newly discovered vulnerabilities.
- Purge old container images from registries. These images are built from older versions of Linux and application packages and are a nuisance at best, and at worst, accidentally deployed with critical, easily exploitable vulnerabilities. Create a well-defined policy for what needs to be saved, for how long, and who should have access, and delete everything else that is past its useful life.

SBoM

Similar to the list of ingredients on food packaging, a Software Bills of Materials (SBoM) provides details on the different components included within a supplied product: open source dependencies, containers, and build tools used. SPDX aims to standardize how SBoMs are defined, and provides a machine-readable format the software industry can use to build tooling that helps overcome security challenges. OWASP CycloneDX is a lightweight SBoM standard designed for use in application security contexts and supply chain component analysis. Following President Biden's Executive Order, SBoMs will be a requirement for any vendor supplying software to the Federal Government and will likely become a cross-industry standard.

Key recommendations:

- Generate and maintain an SBoM to track your third-party dependencies, tools and sources.
- Routinely scan the SBoM for security risks (there is no point in producing the report without actually scrutinizing it).
- Require an SBoM from third-party vendors before or during procurement of new software.

Custom code

While SCA tools can help organizations identify known vulnerabilities in open source packages, they cannot help detect potentially exploitable flaws in custom code.

Custom code can be thought of as a layer on top of the other components in the software supply chain. It also makes the application unique. The underlying mixture of software supply chain components generally defines the sources of data used by custom code, as well as the sanitation and sink functions. For example, using a web framework takes away the need to program complex client-server communication and provides user input as a variable for custom code.

Static Application Security Testing (SAST) tools check custom code for security issues. Often, these issues result from insufficient usage of library functions, ignoring provided security functions, or using deprecated and dangerous functions. Using a SAST tool therefore informs you of the risks resulting from the combination of supply chain components and your custom code.

SAST tools have a reputation for breaking the agile DevOps process by taking long scan times and providing not-actionable or incorrect suggestions. This unbalances supply chain resilience by hindering the business. Modern tools like Snyk Code have changed this model, and can be used early in the software development life cycle.

Key recommendations:

- Use modern SAST tools to scan custom code, as early as possible in the development process.
- Continuously scan custom code during later stages of development.
- Automate scanning process as much as possible.
- Actively manage dependencies and try to reduce complexities.
- Train and enable developers on secure coding best practices.

Infrastructure as code

As the modern software supply chain becomes more complex, development teams bear increasing responsibilities when configuring cloud native applications: containers, infrastructure provisioning, service meshes, and more. Developers do not always have the expertise needed to implement proper security controls for an application, container, infrastructure, or any other software component. This can lead to misconfigurations. Insecure IaC configurations can result in insecure cloud environments, allowing attackers to access applications and privileged data stored in the cloud. Using early onset security scanning, organizations can detect and remediate security misconfigurations in their source code and configuration files.

Key recommendations:

- Fix issues early by using an IaC security tool to scan configuration files and detect vulnerabilities during development in the IDE, CI, and Git Repos.
- Enforce best practice security policies and compliance rulesets throughout the development cycle. Establish guardrails to prevent untagged resources or files defining overly-public access privileges from reaching production.
- Automate weekly scanning of applications for security misconfigurations in infrastructure as code.
- Monitor for configuration drift or differences between the real-time state of your infrastructure and the one defined in your IaC templates, and apply fixes where changes have introduced risk.

Securing your pipelines

If code is the raw material used to build software, development workflows represent the tools, processes, and methodologies that transform this raw material into a product. Software supply chain attacks increasingly target these workflows, leveraging the fact that they are as diverse and complex as the applications they are designed to produce.

The Codecov attack is a good example of how development pipelines themselves can be targeted. In April 2021, Codecov, a company providing software for code coverage and testing tools, reported that attackers had managed to obtain valid credentials from a Docker image and use them to compromise an "upload bash script" used by Codecov customers. Once Codecov's customers downloaded and executed this script, the attackers were able to exfiltrate sensitive information that allowed them to access the customers' resources. The attackers were able to obtain these credentials due to an error in how those Docker images were created.

Build pipelines

In a traditional manufacturing supply chain, the assembly line compiles materials into a finished product. In the software supply chain, the build process is responsible for pulling various code components into a software build. This software assembly line is complex in itself, using dedicated processes, infrastructure, and tools that can be (and have already been) leveraged in a software supply chain attack.

For example: in October 2021, a security issue was discovered in GitHub allowing attackers to use GitHub Actions to bypass required reviews and push unreviewed and malicious code to a protected branch. GitHub has since fixed this issue, but this illustrates the importance of securing build pipelines.

Organizations must aspire to ensure the integrity of all aspects of build pipelines. This starts with the pipelines themselves: build steps, build sources, and build outputs. But it also requires careful examination of the tools and infrastructure used.

Key recommendations:

- Start by mapping out your build pipelines to identify their construction and key components requiring control.
- Use approved and vetted components in the build process, such as hardened container images and artifacts, and checksum-verified tools.
- Pipeline steps should be checked to ensure that the security testing they include is effective.
- Use Reproducible Builds to ensure that vulnerabilities are not introduced during compilation.
- Secure the pipeline infrastructure and configuration.

Source code management

Today, source code management systems (SCM) act as the central hub for an organization's software development lifecycle. By compromising a code repository, attackers can gain access to the software's source code, modify CI/CD pipeline configurations, inject malicious code, hijack credentials, or even provision vulnerable infrastructure. SCMs are therefore another key link in the modern software supply chain that must be addressed. Modern SCMs provide specialized features and configuration settings such as access policy controls and branch protection that can be leveraged to harden security. These mechanisms are not always enabled by default and must be explicitly set.

Key recommendations:

- Allow changes only via pull requests.
- Require code reviews and forbid "push to master" which often automatically triggers a build and release.
- Pay special attention to changes in build scripts and infrastructure configurations.
- Require signed commits to ensure the integrity of source code.
- Use multi-factor authentication (MFA) at the source code repository level for any software project.
- Use and rotate SSH keys to authenticate and access source code.

Secrets and credentials

Software development workflows today use different types of privileged credentials to control access to applications, scripts, tools, and other sensitive information. These could be passwords, encryption keys, SSH keys, API tokens, and so on. As described in the section on the Codecov attack, when these credentials are exposed, they can be used by attackers to compromise the security of an organization's software and, subsequently, the security of the software's end users.

Key recommendations:

- Use encrypted secrets for safe credential storage.
- Secrets should not be committed to a source control repository.
- Use a secrets management tool to store and encrypt secrets, enforce access controls, and manage secrets.
- Scan your source code repositories to ensure secrets are not committed by mistake.
- Use automated service account rotation for credentials.
- Assign restrictive permissions to tokens to ensure a minimal blast radius.

Third-party tools

The open source packages used to build applications are not the only third-party components in the software supply chain. Third-party software provided by software vendors, partners, and service providers can also introduce risk. President Biden's Executive Order calls for NIST to formulate guidelines to help the Federal Government evaluate providers' supply chain security posture, such as verifying adherence to standards and certifications and requiring an SBoM (see SBOM section above).

Key recommendations:

- Require third-party software vendors to be certified against relevant security standards.
- Require an SBoM from third-party software vendors before or during procurement of new software.

DevSecOps

DevSecOps refers to the integration of security practices into a DevOps software delivery model. Its foundation is a culture where development and operations are enabled through process and tooling to take part in a shared responsibility for delivering secure software.

DevSecOps has many benefits, including faster delivery, reduced costs, overall greater business success, and a stronger security posture. The key elements of the model align well with a software supply chain security framework, such as integrating security as early as possible during and throughout software development, and automating security testing and validation to minimize human error. In the context of software supply chain security, DevSecOps also reduces the likelihood of an attacker compromising a development pipeline.

It is not surprising that section 4 in President Biden's Executive Order calls out the use of automated tools and processes for enhancing supply chain security. Integrating security testing into automated development processes (such as CI/CD) ensures continuous security across the build, deploy, and release stages.

Similar to DevOps, DevSecOps is a long-term, cross-team, and continuous effort. It cannot be achieved without a deep change in the culture of the organization. To tighten your software supply chain security, however, there are some key DevSecOps practices that can be adopted early on.

Key recommendations:

- Automate security as part of your CI/CD pipelines.
- Align with key stakeholders in the organization, such as the DevOps and software development teams.
- Create a Security Champions program to bring developers into the fold.
- Empower developers to take more responsibility for security with developer-friendly security tooling.
- Provide developers with guardrails, guidance and training in the form of a governance framework:
  - Maintain an inventory of dependencies, pipelines and contributors
  - Define security policies to manage risk explicitly
  - Policies should be reasonable and maintain the balance between the need for security and fast-paced development
  - Ensure that these policies are applied automatically during all stages of development

For more information about DevSecOps, please refer to our DevSecOps Hub.

Summary

There is no reason to believe that the phenomenon of software supply chain attacks will disappear any time soon. On the contrary, these attacks will most likely increase in both frequency and complexity, affecting more and more organizations and exacting a growing cost.

That's the bad news.

The good news is that the security community, backed by government and corporations, is rising to the challenge. A set of best practices is emerging to help organizations move towards a framework for software supply chain security.

These guidelines are still very much in flux and can appear to be too comprehensive. Rome was not built in a day. Security leaders should work together with other stakeholders in the software development process to address the essentials. First: take measures to secure the code making up applications with dedicated application security testing tools such as SCA and SAST. Second: secure development processes and the development environment. Adopting a DevSecOps model can help organizations implement these two first steps with principles such as developer enablement and security automation.