# How Software Engineering Leaders Can Mitigate Software Supply Chain Security Risks

Refreshed 22 November 2022, Published 15 July 2021 - ID G00752454 - 19 min read

FOUNDATIONAL This research is reviewed periodically for accuracy.

By Analyst(s): Manjunath Bhat, Dale Gardner, Mark Horvath

Initiatives: Software Engineering Practices; Build a World-Class Software Engineering Organization; Security of Applications and Data; Software Engineering Technologies

Attackers are targeting software development systems, open-source artifacts and DevOps pipelines to compromise software supply chains. Software engineering leaders must guide their teams to protect the integrity of the software delivery process by adopting practices described in this research.

## Overview

### Key Findings

- The increased threats of malicious code injection makes it critical to protect internal code and external dependencies (both open-source and commercial).
- Leaked secrets or other sensitive data and code tampering prior to release are consequences of a compromised software build and delivery pipeline.
- Failure to enforce least privilege entitlements and flat network architectures enables attackers to move laterally against the operating environment, putting the enterprise at greater risk.

### Recommendations

Software engineering leaders focused on software engineering strategies should work with their security and risk counterparts to:

- Protect the integrity of internal and external code by enforcing strong version-control policies, using artifact repositories for trusted content, and managing vendor risk throughout the delivery life cycle.

- Harden the software delivery pipeline by configuring security controls in CI/CD tools, securing secrets and signing code and container images.
- Secure the operating environment for software engineers by governing access to resources using principles of least privilege and a zero-trust security model.

## Strategic Planning Assumption

By 2025, 45% of organizations worldwide will have experienced attacks on their software supply chains, a three-fold increase from 2021.

## Introduction

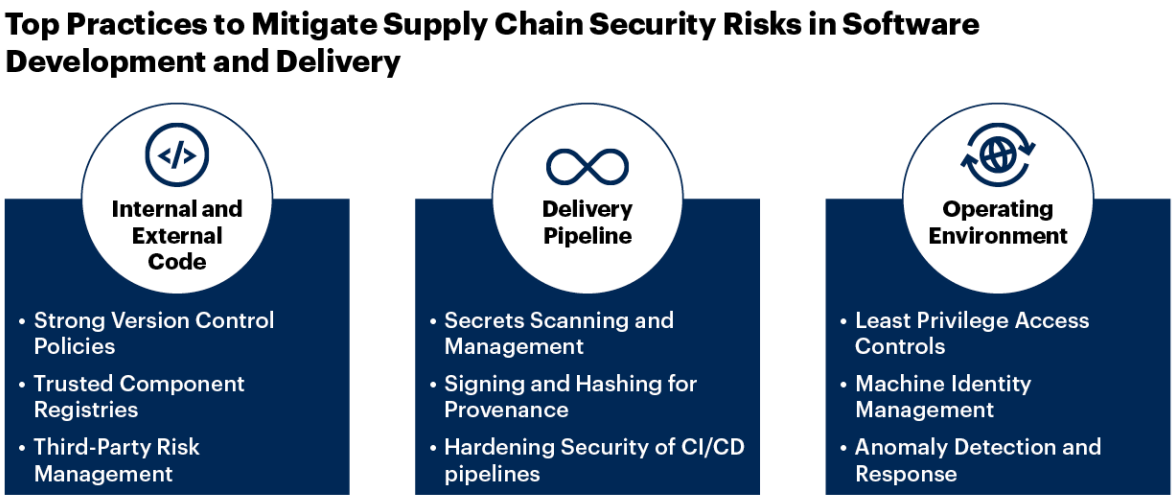
Software engineering leaders are at the forefront of digital business innovation. They are responsible not only for software development and delivery, but are also increasingly accountable for implementing security practices. These practices have traditionally focused on activities such as scanning code for potential security vulnerabilities and patching software systems.

However, software supply chain attacks are becoming increasingly sophisticated, with malicious actors exploiting weaknesses at every stage in the software procurement, development and delivery life cycle. This includes everything from injecting malicious code into open-source packages to installing back doors in postdeployment software updates.

As a result, software engineering teams must assume that all code (both externally sourced and internally developed), development environments and tooling may have been compromised. In addition, security hygiene should now extend to external code dependencies and commercial off the-shelf (COTS) software, which includes the use of third-party APIs.

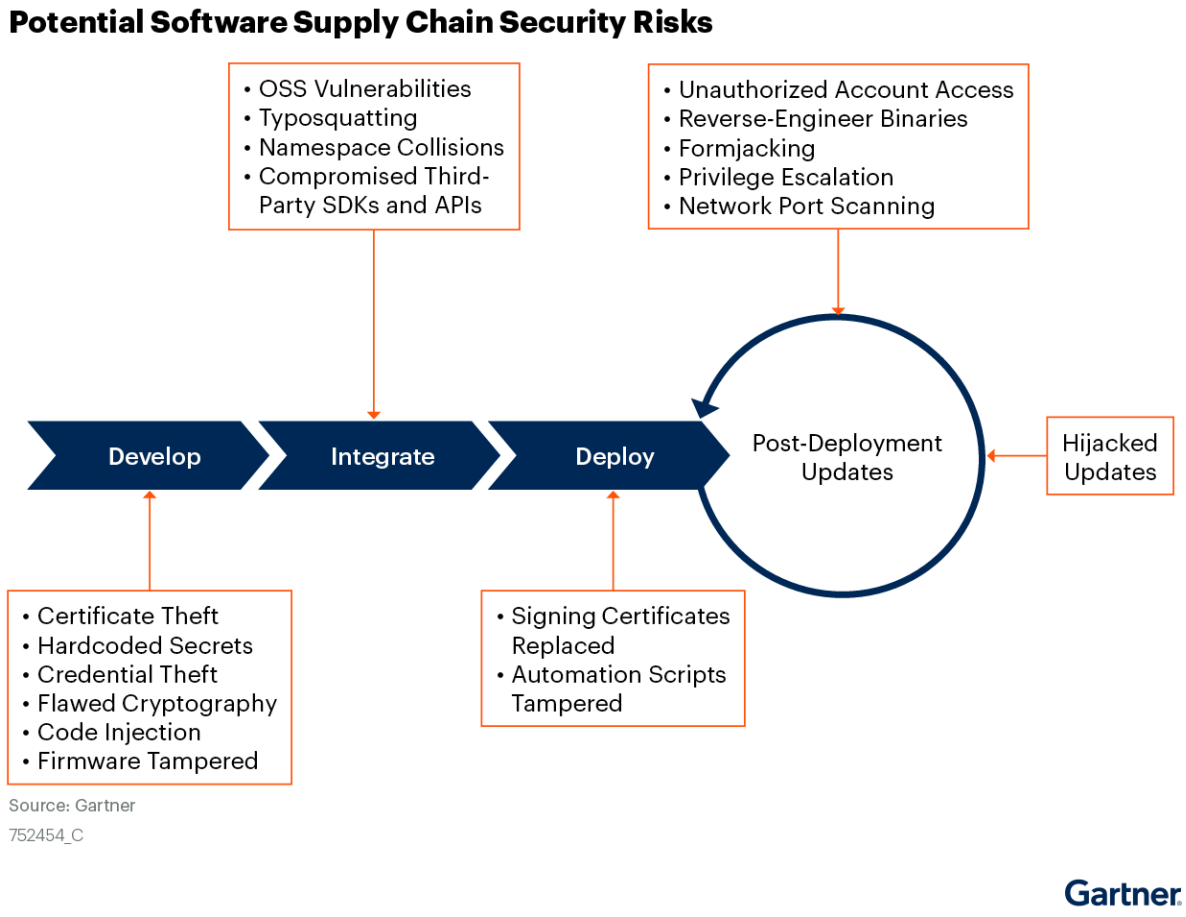
This research explains how software engineering leaders can counter the threat of software supply chain attacks. See Figure 1 for secure development practices to guard against software supply chain attacks. Figure 2 highlights the potential security risks at each stage of the delivery process.

Figure 1: Top Practices to Mitigate Supply Chain Security Risks in Software Development and Delivery




Source: Gartner
752454_C

## Figure 2: Potential Software Supply Chain Security Risks




*A software supply chain attack is the act of compromising software or one of its dependencies at any stage throughout its development, delivery and usage. Although the precise attack vector may vary in each case, the attacker usually gains unauthorized access to development environments and infrastructure including version control systems, artifact registries, open-source repositories, continuous integration pipelines, build servers or application servers. This allows the attacker to modify source code, scripts and packages, and establish back doors to steal data from the victim's environment. Attacks are not limited to external actors; they can come from insider threats as well.*

*The attacks on SolarWinds (2020), NetBeans IDE (2020), Kaseya (2021) and Codecov (2021) represent four prominent examples of software supply chain attacks (see the Evidence section and Notes 1 through 5). Gartner believes that By 2025, 45% of organizations worldwide will have experienced attacks on their software supply chains, a three-fold increase from 2021.*

*This research will address the following security concerns Gartner encounters in client inquiries:*

- *Compromise of continuous integration/continuous delivery (CI/CD) systems*
- *Injection of malware into legitimate software*
- *Inclusion of vulnerable and malicious dependencies*

---

# Analysis

## Protect the Integrity of Internal and External Source Code

Software engineering teams use version control systems (VCSs) and artifact repositories to maintain internally developed code and external artifacts. Failing to enforce security controls in these tools exposes source code and artifacts to potential manipulation and tampering.

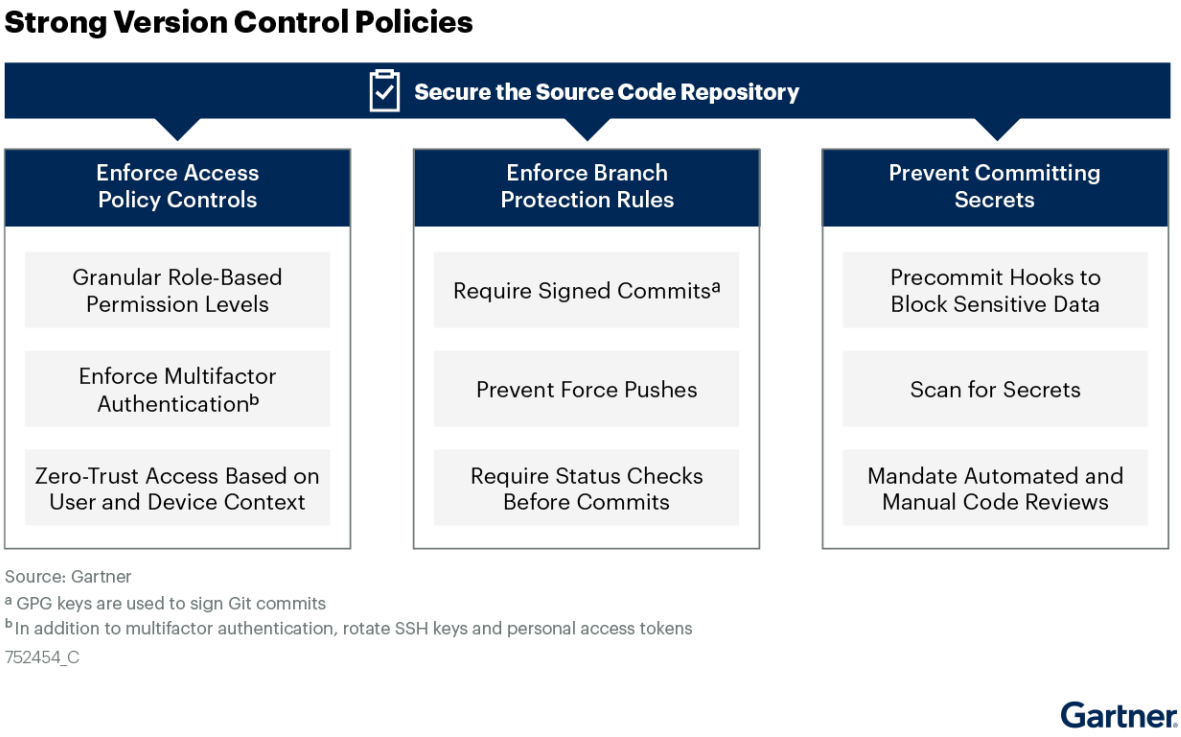
We recommend three practices that protect the integrity of internal and external code:

1. Strong version control policies
2. Trusted component registries
3. Third-party risk management

### Strong Version Control Policies

Git-based VCSs including BitBucket, GitHub and GitLab provide native security and access protection capabilities. Software engineering teams must leverage access policy controls, branch protection and secrets scanning capabilities. These controls are not enabled by default and must be explicitly set. See Figure 3.

Figure 3: Strong Version Control Policies




Secrets and credentials should never be stored in source code repositories, but software engineers can accidentally commit secrets to source control. Since any user who has access to the repository can clone the repository and store it anywhere, the cloned repository becomes a treasure trove for attackers looking to steal credentials, API keys or secrets. We recommend continuous scanning of repositories to check for files embedded with secrets using either open-source tools or provider-native capabilities (see Table 1).

Table 1: Representative list of Secrets Scanning tools for Git Repositories

| Open-Source Tools | Vendors |
|---|---|
| git-secrets: Open sourced by AWS Labs | GitHub Secrets Scanning |
| Repo Supervisor: Open sourced by Auth0 | GitLab Secret Detection |
| truffleHog: Searches for secrets in Git repos | Bitbucket Secrets Scan |
| Gitleaks: Scans repos and commits for secrets | GitGuardian |
| Deadshot: Open sourced by Twilio | SpectralOps |
| | |

Source: Gartner

### Trusted Component Registries

As software is increasingly assembled using open-source components, third-party packages and public APIs, the threat of supply chain attacks due to malicious code looms large (see Note 5 for examples of package typosquatting attacks in popular open-source libraries). We recommend the use of artifact (and container) repositories, software composition analysis tools and code scanning tools.

### Artifact Repositories

Artifact repositories enable securing, versioning and safely distributing software packages — both internally built and externally sourced. The repositories act as a trusted source for sanctioned and vetted artifacts and software components. This enables centralized governance, visibility, auditability and traceability into software "ingredients."

Since the repositories can act as proxies to external public registries, it has the added benefit of keeping the packages continuously updated and patched. One of the defense agencies uses a centralized artifact repository ("Iron Bank" [1]) that stores signed container images for both OSS and COTS. See Note 2 for open-source container signing tools.

### Examples of artifact repositories:

- Azure Artifacts

- AWS CodeArtifact
- GitHub
- GitLab
- Google Artifact Registry
- JFrog Artifactory
- Sonatype Nexus Repository
- Tidelift Catalogs

**Examples of container registries:**

- Amazon ECR
- Azure Container Registry
- CNCF Harbor
- Docker Trusted Registry
- GitHub
- GitLab
- Google Container Registry
- JFrog Artifactory
- Red Hat Quay

#### Software Composition Analysis (SCA)

SCA complements artifact and container registries by analyzing stored artifacts or container images to uncover known vulnerabilities. Without SCA, it is difficult to manage dependencies at scale and to identify known vulnerabilities in published components. Gartner recommends pairing artifact repositories with integrated SCA and open-source governance capabilities. Examples include Sonatype Nexus Repository with IQ Server or JFrog Artifactory with Xray (see Market Guide for Software Composition Analysis).

#### Code Scanning

Although SCA uncovers known vulnerabilities (often CVE IDs) in published software packages, it will not help with detecting potentially exploitable flaws that exist in custom application code. We recommend using application security testing tools for static (SAST), dynamic (DAST) and fuzz testing of application code. Some AST tools offer SCA capabilities (see Magic Quadrant for Application Security Testing).

### Third-Party Risk Management

Software engineering teams not only build their own software but also consume software developed by other organizations (including vendors, partners and service providers). This section will focus on assessing and managing the two kinds of supply chain risks typically associated with third-party software:

- Risks due to known vulnerabilities in third-party or open-source dependencies (for example, Equifax, SaltStack). [3,4]
- Risks due to back doors/malware implanted in externally procured software (for example, SolarWinds attacks). [5]

Gartner recommends the following practices to mitigate these risks:

**Check for adherence to standards and certifications:** Require software suppliers to be certified against relevant security standards such as UL 2900 for IoT certification and ISO/IEC 27034 to ensure adherence to consistent and formalized application security practices. This may include requiring a specified level of developer testing and evaluation. For example, static and dynamic code analysis, threat modeling and vulnerability analysis, third-party verification of processes, manual code review and penetration testing.

See Note 6 for frameworks and standards that help evaluate your provider/partner's supply chain security posture.

**Audit the provider's software build, deployment and upgrade process:** Ask these questions to benchmark software providers against a minimal baseline:

- Does the provider have the necessary controls to secure their SDLC process? (see 12 Things to Get Right for Successful DevSecOps.)
- What process does the provider follow to patch its own software and its dependencies? Request a software bill of materials that helps track nested dependencies. See Note 3 for tracking open-source dependency chains and assessing security posture of open source projects (upstream).

- Is the mechanism to deliver the patch protected from external attacks?
- What is the SLA for patching a vulnerability discovered in the vendor's software or its dependencies?

> **In software, the chain isn't as strong as its weakest link; it's as weak as all the weak links multiplied together.**
>
> — *Steve McConnell*

## Harden the Software Development and Delivery Pipeline

Gartner recommends three practices to strengthen the security of the software delivery pipelines:

- Implement secrets management
- Implement signing and hashing to verify integrity of source code
- Configure security controls in CI/CD pipelines

### Implement Secrets Management

Hard-coding secrets in code and configuration files significantly increases the risk of compromising build pipelines and development environments. In addition, storing any type of secret in a container image can expose that data inadvertently, particularly if images are stored in public registries. Software engineering teams must continuously check committed code and artifacts for embedded secrets.

Secrets management provides a disciplined approach to managing and securing secrets such as credentials, passwords, API tokens and certificates. We recommend the use of secrets management tools to automate creation, storage, retrieval and revocation of secrets. This helps avoid embedding (hard-coding) secrets in source code, configuration files and infrastructure automation scripts. See Table 2 for representative providers of secrets management tools for multiple development scenarios.

## Table 2: Secrets Management Tools

(Enlarged table in Appendix)

| Use Case | Secrets Management Tool |
|---|---|
| Platform-agnostic tools | ■ Akeyless<br>■ CyberArk Conjur<br>■ Thycotic Secrets Vault<br>■ HashiCorp Vault |
| Cloud-provider tools | ■ AWS Secrets Manager<br>■ Azure Key Vault<br>■ GCP Secret Manager |
| Container-native environments | ■ Kubernetes Secrets (etcd),<br>■ Sealed Secrets |
| Configuration Management | ■ Ansible Vault<br>■ Chef Data Bag<br>■ Puppet Hiera |

Source: Gartner

**Secrets such as credential files, private keys, passwords and API tokens should not be committed to a source control repository. Use a secrets management tool to securely store and encrypt secrets, enforce access controls and manage secrets (that is, create, rotate and revoke).**

### Implement Signing and Hashing to Verify Integrity of Source Code

Hashing and signing can be used to verify integrity of source code and binaries. VCSs generate hashes (unique identifiers) for individual files during commits. These hashes help validate that the files are not altered in transit. Likewise, compilers generate hashes as well. Compiler-generated hashes (during CI/CD) can be compared with file hashes generated by static file analyzers (during scanning). This ensures that the code being shipped is the same as the code that was scanned. See Note 7 for hashing and code signing tools.

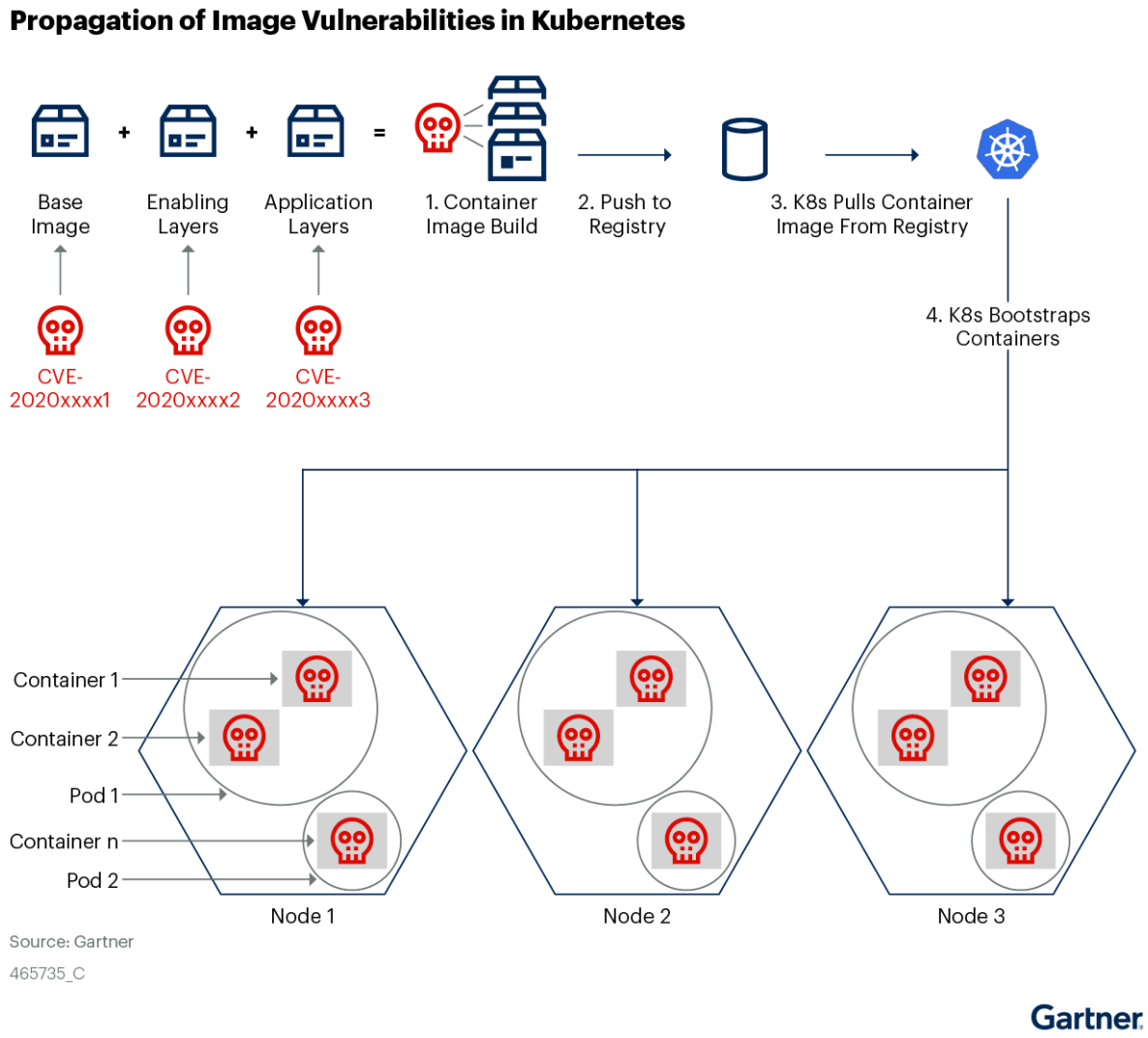
### Commit Signing

Hashing does not address the needs of provenance and authenticity, which is why we recommend signing. VCSs support signed commits to provide others the assurance that the files originated from a trusted source. As examples, GitHub and GitLab attach a "verified" label against signed commits when the signature can be verified for valid users. Think of this like Twitter verified accounts — where Twitter confirms the account holder's identity.

### Container Signing

As organizations move to container-based deployments and source containers from different locations, ensuring the integrity of container images becomes paramount. Gartner recommends signing container images even if your organization builds and maintains internal images. This is because any issue in third-party code or dependencies impacts the security posture of your running applications(see Figure 4). See Note 2 for open-source container signing tools.

Figure 4: Propagation of Container Image Vulnerabilities in Kubernetes




## Configure Security Controls in CI/CD Pipelines

CI/CD systems, if left unsecured, introduce security risks in the software delivery pipeline. For example, attackers can manipulate build pipeline definitions to suppress checks, allowing malicious code to pass through or redirect releases to a malicious delivery target. CI/CD pipelines must be configured with the elevated security and access controls, as they may be turned off by default.

Attackers are increasingly targeting build pipelines as an attack vector. Therefore, software engineering leaders must implement security tools to protect code integrity and prevent code tampering in the build pipeline. Representative providers of these tools include Apiiro, Argon, Cycode, Garantir, GrammaTech, JFrog (Vdoo) and RunSafe Security.

Software engineering teams must adopt these practices to protect their CI/CD pipelines:

- **Reproducible build practices** to ensure that a given source code always results in the same build output. See Note 4 for details on the “reproducible-builds” project and tools.
- **Signed Pipelines** that create immutable, verifiable artifacts. For example, JFrog Signed Pipelines and Tekton Chains enable signing artifacts generated during a pipeline run to ensure immutability and verify provenance at the end of the pipeline execution.

---

*Compromised IDEs resulting in trojanized builds present the most serious supply chain security risk in software delivery. Examples include XcodeGhost malware (Xcode IDE in 2015), Octopus Scanner (NetBeans IDE in 2020), vulnerable VS Code extensions and Node.js Debugger.*

*Browser-based IDEs eliminate the risk of compromising locally installed IDEs on developer machines. Browser-based IDEs either enable web access to a remote development environment or sandbox the IDE within the browser security context. This decouples the development workspace from the physical workstation which may not be adequately protected. Examples of browser-based IDEs include Codeanywhere, GitHub Codespaces, Gitpod, Replit and StackBlitz.*

---

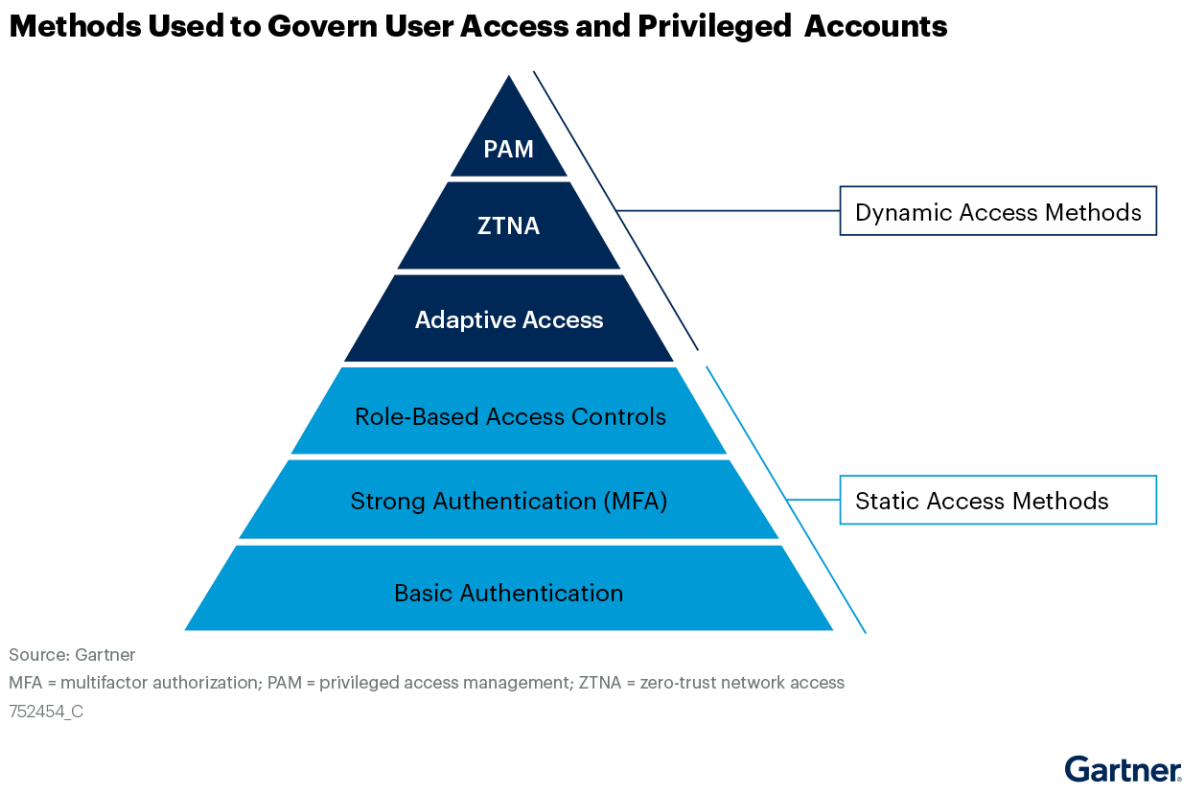
## Secure the Operating Environment for Developers

Software development environments span multiple distributed systems, platforms and tools, communicating with each other in the software delivery life cycle using privileged service accounts. For example, build machines communicate with source code repositories to pull source code and artifact repositories to pull common packages, and connect to deployment targets to deploy binaries. The risk is that the tools and services default to running with elevated system privileges without privilege access controls in place.

### Least Privilege Access Policies and Methods

The ability to connect to different machines on a network and elevated system privileges allows attackers to infiltrate other machines and services once they gain access to one system. In addition, a compromised executable can establish unauthorized connections to other critical systems unless the right access controls are put in place. Therefore, we recommend the use of role-based authentication and authorization, adaptive access using zero-trust security model and privilege access management (see Figure 5).

Figure 5: Methods to Govern User Access and Privileged Accounts




*Zero-Trust Network Access (ZTNA) provides controlled identity- and context-aware access to development resources, reducing the surface area for supply chain attacks. It eliminates excessive implicit trust placed on machines and services simply because they share the same network, replacing it with explicit identity-based trust (see Market Guide for Zero Trust Network Access).*

We recommend privileged access management (PAM) tools to monitor and protect privileged service accounts that run builds and automation jobs. In the SolarWinds attack, attackers modified trusted domains and abused privileged roles (see Note 1). We see that as a confirmation that privileged accounts are a primary target. Mitigating this risk often requires privileged accounts to be managed by a PAM (see Top 10 Lessons Learned From the SolarWinds Attack).

PAM tools help vault privileged passwords, limit access to authorized users, rotate credentials frequently and monitor the usage of privileged accounts. Vendors such as Akeyless (Zero Trust Access), Britive and HashiCorp (Boundary) combine dynamic secrets, authentication, authorization and just-in-time credentials to enforce least privilege access to services and resources.

### Machine Identity Management

Distributed application architectures, cloud-native infrastructure and APIs-as-products have increased the granularity and volume of machine identities. Hosts, containers, VMs, applications, database servers, services and API endpoints all use distinct machine identities. Machine identities enable the services and endpoints to uniquely authenticate themselves while interacting with other services.

**The scale and speed at which machine identities are used to authenticate and access services throughout the software supply chain makes machine identity management now an imperative.**

Examples of machine identities include server credentials such as TLS certificates and SSH host keys, and client credentials such as OAuth credentials, API tokens and database connection strings. Machine identity management (MIM) is the discipline of managing the life cycle and access to these identities. MIM is not a single tool or market. It is a collection of practices and tools that enhances trust and integrity of machine-to-machine interactions, which is critical to securing development environments (see Table 3).

## Table 3: Tools and Providers for Machine Identity Management

(Enlarged table in Appendix)

| Use Case | Domain | Sample Providers |
|---|---|---|
| Encryption of data at rest, management of symmetric keys | Key management systems | ■ Akeyless<br>■ AWS<br>■ KMS<br>■ Azure<br>■ Twilio (Ionic)<br>■ Fortanix<br>■ PKWARE<br>■ Thales and Townsend Security |
| Storing secrets used in the DevOps pipeline, issuance of machine identities to containers | Secrets management | ■ Akeyless<br>■ AWS<br>■ Microsoft Azure<br>■ BeyondTrust<br>■ CyberArk<br>■ Fortanix<br>■ Google Cloud Platform (GCP)<br>■ HashiCorp<br>■ ThycoticCentrify |
| Authentication, encryption and signatures for code signing | PKI and certificate management | ■ AppViewX<br>■ AWS<br>■ DigiCert<br>■ Entrust<br>■ GlobalSign<br>■ Keyfactor<br>■ Microsoft<br>■ The Nexus Group<br>■ Sectigo<br>■ Venafi |
| Discovering and controlling privileged access to critical systems | Privileged access management | ■ Akeyless<br>■ BeyondTrust<br>■ Broadcom<br>■ CyberArk<br>■ One Identity<br>■ ThycoticCentrify |

Source: Gartner

### Anomaly Detection and Automated Response

One of the lessons from the SolarWinds attack is that software engineering teams are ill-equipped to detect and respond to anomalies before damage is done. Malicious attacks on software development pipelines have a high likelihood of surfacing as an anomalous activity.

Examples of such anomalous activity include:

- Executables establishing seemingly strange and unnecessary connections with their “command and control” centers.

- Increases in the number of processes or threads, CPU and memory utilization on select machines.
- Spikes in network access, repository uploads/downloads and unexpected directory access traffic.
- Monitor for cloning of source code repositories using logging or other monitoring tools, such as an EPP or SIEM. CASBs can also be useful in cases of SaaS-based version control systems.

Software engineering leaders must work closely with security and risk teams to understand and define the expected behavior of their development platforms and tools so they can detect anomalies in real time. For example, EDR, CWPP, NDR or tools such as osquery can monitor for system anomalies. Build systems, including PCs used by software engineers, should not be exempt from EPP/EDR protection for this reason.

Anomaly detection and response is especially critical in container-native, GitOps-based deployments that automate the full code-to-container workflow. Although container image scanning tools in the development phase help detect known vulnerabilities, software engineering teams must deploy tools to visualize container traffic, identify cluster misconfigurations and alert on anomalous container behavior and security events.

See Market Guide for Cloud Workload Protection Platforms for container and Kubernetes-focused tools.

## Evidence

[1] Iron Bank

[2] 2020 State of the Software Supply Chain Report, Sonatype

[3] Equifax Dat Breach, Epic.org

[4] SaltStack Authorization Bypass, F-Secure Labs

[5] SolarWinds Security Advisory, SolarWinds

## Note 1: Examples of Software Supply Chain Security Attacks

SolarWinds — Roughly 18,000 customers of SolarWinds downloaded trojanized versions of its Orion IT monitoring and management software. The supply chain attack led to the breach of government and high-profile companies after attackers deployed a backdoor dubbed SUNBURST or Solorigate.

Octopus Scanner Malware — Octopus Scanner, an OSS supply chain malware targets NetBeans IDE injecting backdoor code into resulting JAR files built with the IDE.

Codecov — U.S. cybersecurity firm Rapid7 had also revealed that some of their source code repositories and credentials were accessed by Codecov attackers.

## Note 2: Open-Source Projects for Container Signing and Admission Tools

Grafeas defines an API spec for managing metadata about software resources, such as container images, VM images, JAR files, and scripts. It provides a centralized knowledge base for artifacts, their origins, vulnerabilities, dependencies, etc. that make up the code-to-container supply chain.

Kritis is a Kubernetes admission controller that runs policy checks defined by the Kubernetes cluster admin, at runtime and then either approves or denies the pod to be launched — based on vulnerabilities in the image or if the image is not obtained from a trusted source.

Kritis Signer is a command-line tool that creates attestations for a container image.

Cosign signs container images. Cosign is developed as part of the sigstore project hosted by the Linux Foundation. The goal of sigstore is to ease the adoption of signing software artifacts.

## Note 3: Tracking Open-Source Dependency Chains

Open Source Insights shows information about a package without requiring users to install the package first. Developers can see what installing a package means for their project, how popular it is, find links to source code, and then decide whether it should be installed.

OSSF Scorecard is a tool that helps evaluate security posture of OSS projects.

Supply chain Levels for Software Artifacts (SLSA, pronounced “salsa”), an end-to-end framework for ensuring the integrity of software artifacts throughout the software supply chain. It is inspired by Google’s internal “Binary Authorization for Borg” and secures all of Google’s production workloads.

## Note 4: Reproducible Builds

The motivation behind the Reproducible Builds project is to allow verification that no vulnerabilities or backdoors have been introduced during the compilation process. Tools such as diffoscope help compare files, directories and identify what makes them different. It can compare jars, tarballs, ISO images, or PDFs.

Three guiding principles underpin the idea of reproducible builds:

- Deterministic builds: A given source must always yield the same output.
- Hardened build tools: The tools in the build pipeline are hardened and immutable.
- Verifiable output: Ability to detect and fix the discrepancy between expected and actual builds.

## Note 5: Package Typosquatting Supply Chain Attacks

Typosquatting is a type of software supply chain attack where the attacker tries to mimic the name of an existing package in a public registry hoping that developers will accidentally download the malicious package instead of the legitimate one. An npm package (loadsh) was typosquatting the popular lodash package using the transposition of the “a” and “d” characters. See this whitepaper for more details: SpellBound: Defending Against Package Typosquatting.Using artifact registries, software composition analysis and code scanning reduces the risk from package typosquatting attacks on public repositories.

As noted in the 2020 State of the Software Supply Chain Report [2] by Sonatype, *“Bad actors are no longer waiting for public vulnerability disclosures. Instead, they are taking the initiative and actively injecting malicious code into open-source projects that feed the global supply chain. By shifting their focus “upstream,” bad actors can infect a single component, which will then be distributed “downstream” using legitimate software workflows and update mechanisms.”*

ReversingLabs revealed over 700 malicious gems (packages written in Ruby programming language) being distributed through the RubyGems repository. Twelve Python libraries uploaded on the official Python Package Index (PyPI) contained malicious code. In Jan 2021, Sonatype discovered three malicious packages that were published to npm repository, all of which leveraged brandjacking and typosquatting techniques.

## Note 6: Frameworks and Standards for Evaluating Supply Chain Security

Evaluating Your Supply Chain Security — a Checklist by Cloud Native Computing Foundation (CNCF)

NIST Secure Software Development Framework

NIST, Security and Privacy Controls for Information Systems and Organizations

Ul 2900 for IoT Certification

ISO/IEC 27034

## Note 7: Sample Hashing and Signing Tools

Visual Studio — Hashing Source Code Files with Visual Studio to Assure File Integrity

GaraSign For Code Signing

---

## Recommended by the Authors

Some documents may not be available as part of your current Gartner subscription.

Top Security and Risk Management Trends 2021

Software Engineering Strategies Primer for 2021

Software Engineering Technologies Primer for 2021

Magic Quadrant for Application Security Testing

Guidance Framework for Securing Kubernetes

Solution Path for Continuous Delivery With DevOps

3 Steps to Integrate Security Into DevOps

## Table 1: Representative list of Secrets Scanning tools for Git Repositories

| Open-Source Tools | Vendors |
|---|---|
| git-secrets: Open sourced by AWS Labs | GitHub Secrets Scanning |
| Repo Supervisor: Open sourced by Auth0 | GitLab Secret Detection |
| truffleHog: Searches for secrets in Git repos | Bitbucket Secrets Scan |
| Gitleaks: Scans repos and commits for secrets | GitGuardian |
| Deadshot: Open sourced by Twilio | SpectralOps |

Source: Gartner

## Table 2: Secrets Management Tools

| *Use Case* | *Secrets Management Tool* |
|---|---|
| Platform-agnostic tools | ■ Akeyless<br>■ CyberArk Conjur<br>■ Thycotic Secrets Vault<br>■ HashiCorp Vault |
| Cloud-provider tools | ■ AWS Secrets Manager<br>■ Azure Key Vault<br>■ GCP Secret Manager |
| Container-native environments | ■ Kubernetes Secrets (etcd),<br>■ Sealed Secrets |

| Use Case | Secrets Management Tool |
| --- | --- |
| Configuration Management | ■ Ansible Vault<br>■ Chef Data Bag<br>■ Puppet Hiera |

Source: Gartner

## Table 3: Tools and Providers for Machine Identity Management

| *Use Case* | *Domain* | *Sample Providers* |
|---|---|---|
| Encryption of data at rest, management of symmetric keys | Key management systems | ■ Akeyless<br>■ AWS<br>■ KMS<br>■ Azure<br>■ Twilio (Ionic)<br>■ Fortanix<br>■ PKWARE<br>■ Thales and Townsend Security |

| *Use Case* | *Domain* | *Sample Providers* |
| --- | --- | --- |
| Storing secrets used in the DevOps pipeline, issuance of machine identities to containers | Secrets management | ■ Akeyless<br>■ AWS<br>■ Microsoft Azure<br>■ BeyondTrust<br>■ CyberArk<br>■ Fortanix<br>■ Google Cloud Platform (GCP)<br>■ HashiCorp<br>■ ThycoticCentrify |

| *Use Case* | *Domain* | *Sample Providers* |
| --- | --- | --- |
| Authentication, encryption and signatures for code signing | PKI and certificate management | ■ AppViewX<br>■ AWS<br>■ DigiCert<br>■ Entrust<br>■ GlobalSign<br>■ Keyfactor<br>■ Microsoft<br>■ The Nexus Group<br>■ Sectigo<br>■ Venafi |
| Discovering and controlling privileged access to critical systems | Privileged access management | ■ Akeyless<br>■ BeyondTrust<br>■ Broadcom<br>■ CyberArk<br>■ One Identity<br>■ ThycoticCentrify |

| Use Case | Domain | Sample Providers |
| --- | --- | --- |

Source: Gartner