A03:2025 — Software Supply Chain Failures

January 6, 2026

Document Summary

This document consolidates the provided content for *A03:2025 — Software Supply Chain Failures* into a structured, print-ready reference, including background context, scoring metrics, vulnerability indicators, prevention guidance (SBOM, dependency tracking, hardening, and change management), attack scenarios, references, and the mapped CWE list.

Contents

1 Background

This was top-ranked in the Top 10 community survey with exactly 50% of respondents ranking it #1. Since initially appearing in the 2013 Top 10 as *"A9 – Using Components with Known Vulnerabilities"*, the risk has grown in scope to include all supply chain failures, not just ones involving known vulnerabilities.

Despite this increased scope, supply chain failures continue to be a challenge to identify, with only 11 Common Vulnerabilities and Exposures (CVEs) having the related CWEs. However, when tested and reported in the contributed data, this category has the highest average incidence rate at 5.19%. The relevant CWEs are CWE-447 (Use of Obsolete Function), CWE-1104 (Use of Unmaintained Third Party Components), CWE-1329 (Reliance on Component That is Not Updateable), and CWE-1395 (Dependency on Vulnerable Third-Party Component).

2 Score Table

Metric	Value
CWEs Mapped	6
Max Incidence Rate	9.56%
Avg Incidence Rate	5.72%
Max Coverage	65.42%
Avg Coverage	27.47%
Avg Weighted Exploit	8.17
Avg Weighted Impact	5.23
Total Occurrences	215,248
Total CVEs	11

Table 1: Provided scoring summary for Software Supply Chain Failures.

3 Description

Software supply chain failures are breakdowns or other compromises in the process of building, distributing, or updating software. They are often caused by vulnerabilities or malicious changes in third-party code, tools, or other dependencies that the system relies on.

3.1 Likely Vulnerability Indicators

You are likely vulnerable if:

- You do not carefully track the versions of all components that you use (both client-side and server-side). This includes components you directly use as well as nested (transitive) dependencies.
- The software is vulnerable, unsupported, or out of date. This includes the OS, web/application server, database management system (DBMS), applications, APIs and all components, runtime environments, and libraries.

- You do not scan for vulnerabilities regularly and subscribe to security bulletins related to the components you use.
- You do not have a change management process or tracking of changes within your supply chain, including tracking IDEs, IDE extensions and updates, changes to your organization's code repository, sandboxes, image and library repositories, the way artifacts are created and stored, etc. Every part of your supply chain should be documented, especially changes.
- You have not hardened every part of your supply chain, with a special focus on access control and the application of least privilege.
- Your supply chain systems do not have any separation of duty. No single person should be able to write code and promote it all the way to production without oversight from another human being.
- Components from untrusted sources, across any part of the tech stack, are used in or can impact production environments.
- You do not fix or upgrade the underlying platform, frameworks, and dependencies in a risk-based, timely fashion. This commonly happens when patching is a monthly or quarterly task under change control, leaving organizations open to days or months of unnecessary exposure before fixing vulnerabilities.
- Software developers do not test the compatibility of updated, upgraded, or patched libraries.
- You do not secure the configurations of every part of your system (see A02:2025 — Security Misconfiguration).
- Your CI/CD pipeline has weaker security than the systems it builds and deploys, especially if it is complex.

4 How to Prevent

4.1 Patch Management and Component Governance

There should be a patch management process in place to:

1. Centrally generate and manage the Software Bill of Materials (SBOM) for the entire software portfolio.
2. Track not just direct dependencies, but their (transitive) dependencies, and so on.
3. Reduce attack surface by removing unused dependencies, unnecessary features, components, files, and documentation.
4. Continuously inventory the versions of both client-side and server-side components (e.g., frameworks, libraries) and their dependencies using tools like OWASP Dependency-Track, OWASP Dependency-Check, retire.js, etc.
5. Continuously monitor vulnerability sources such as Common Vulnerabilities and Exposures (CVE), National Vulnerability Database (NVD), and Open Source Vulnerabilities (OSV) for issues affecting the components you use.
6. Use software composition analysis (SCA), software supply chain, or security-focused SBOM tooling to automate monitoring; subscribe to alerts for security vulnerabilities related to your components.

7. Only obtain components from official (trusted) sources over secure links. Prefer signed packages to reduce the chance of including a modified, malicious component (see A08:2025 — Software and Data Integrity Failures).
8. Deliberately choose which version of a dependency you use and upgrade only when there is need.
9. Monitor for libraries and components that are unmaintained or do not create security patches for older versions. If patching is not possible, consider migrating to an alternative. If that is not possible, consider deploying a virtual patch to monitor, detect, or protect against the discovered issue.
10. Update your CI/CD, IDE, and any other developer tooling regularly.
11. Avoid deploying updates to all systems simultaneously. Use staged rollouts or canary deployments to limit exposure in case a trusted vendor is compromised.

4.2 Change Management and Supply Chain Traceability

There should be a change management process or tracking system in place to track changes to:

- CI/CD settings (all build tools and pipelines)
- Code repositories
- Sandbox areas
- Developer IDEs
- SBOM tooling, and created artifacts
- Logging systems and logs
- Third-party integrations, such as SaaS
- Artifact repositories
- Container registries

4.3 Hardening and Identity Controls

Harden the following systems, including enabling MFA and locking down IAM:

- **Code repositories:** Avoid checking in secrets, protect branches, and maintain backups.
- **Developer workstations:** Regular patching, MFA, monitoring, and related endpoint controls.
- **Build servers & CI/CD:** Separation of duties, access control, signed builds, environment-scoped secrets, tamper-evident logs, and more.
- **Artifacts:** Ensure integrity via provenance, signing, and time stamping; promote artifacts rather than rebuilding for each environment; ensure builds are immutable.
- **Infrastructure as code:** Manage like all code, including pull requests (PRs) and version control.

Lifecycle Requirement

Every organization must ensure an ongoing plan for monitoring, triaging, and applying updates or configuration changes for the lifetime of the application or portfolio.

5 Example Attack Scenarios

5.1 Scenario #1: Trusted Vendor Compromise (Malware in Updates)

A trusted vendor is compromised with malware, leading to your computer systems being compromised when you upgrade. A frequently cited example is the 2019 SolarWinds compromise that led to approximately 18,000 organizations being compromised:

- https://www.npr.org/2021/04/16/985439655/a-worst-nightmare-cyberattack-the-untold-story-of-

5.2 Scenario #2: Conditional Malicious Behavior from a Compromised Vendor

A trusted vendor is compromised such that it behaves maliciously only under a specific condition.

The 2025 Bybit theft of $1.5 billion was caused by a supply chain attack in wallet software that only executed when the target wallet was being used.

5.3 Scenario #3: Self-Propagating Package Worm (2025 "Shai-Hulud")

The Shai-Hulud supply chain attack in 2025 was described as the first successful self-propagating `npm` worm. Attacks seeded malicious versions of popular packages, which used a post-install script to harvest and exfiltrate sensitive data to public GitHub repositories. The malware would also detect `npm` tokens in the victim environment and automatically use them to push malicious versions of any accessible package. The worm reached over 500 package versions before being disrupted by `npm`.

This supply chain attack was advanced, fast-spreading, and damaging; by targeting developer machines it demonstrated that developers themselves are prime targets for supply chain attacks.

5.4 Scenario #4: Component Vulnerabilities Inherit Application Privilege

Components typically run with the same privileges as the application itself, so flaws in any component can result in serious impact. Such flaws can be accidental (e.g., coding error) or intentional (e.g., a backdoor in a component). Some example exploitable component vulnerabilities discovered are:

- **CVE-2017-5638:** A Struts 2 remote code execution vulnerability enabling execution of arbitrary code on the server; it has been blamed for significant breaches.
- **CVE-2021-44228 ("Log4Shell"):** An Apache Log4j remote code execution zero-day vulnerability; it has been blamed for ransomware, cryptomining, and other attack campaigns.

6 References

- OWASP Application Security Verification Standard: V15 Secure Coding and Architecture
- OWASP Cheat Sheet Series: Dependency Graph SBOM
- OWASP Cheat Sheet Series: Vulnerable Dependency Management
- OWASP Dependency-Track
- OWASP CycloneDX
- OWASP Application Security Verification Standard: V1 Architecture, design and threat modelling
- OWASP Dependency-Check (for Java and .NET libraries)

- OWASP Testing Guide — Map Application Architecture (OTG-INFO-010)
- OWASP Virtual Patching Best Practices
- The Unfortunate Reality of Insecure Libraries
- MITRE Common Vulnerabilities and Exposures (CVE) search
- National Vulnerability Database (NVD)
- Retire.js for detecting known vulnerable JavaScript libraries
- GitHub Advisory Database
- Ruby Libraries Security Advisory Database and Tools
- SAFECode Software Integrity Controls (PDF)
- Glassworm supply chain attack
- PhantomRaven supply chain attack campaign

7 List of Mapped CWEs

CWE	Title
CWE-447	Use of Obsolete Function
CWE-1035	2017 Top 10 A9: Using Components with Known Vulnerabilities
CWE-1104	Use of Unmaintained Third Party Components
CWE-1329	Reliance on Component That is Not Updateable
CWE-1357	Reliance on Insufficiently Trustworthy Component
CWE-1395	Dependency on Vulnerable Third-Party Component