

# SOFTWARE SUPPLY CHAIN SECURITY.



Modern software is assembled, not written from scratch — **most of a codebase is open-source dependencies** nobody on the team wrote. Supply chain security tracks what's actually inside that assembly and locks down the pipeline that builds and ships it.

- SBOM
- DEPENDENCY SCANNING
- CODE SIGNING
- PIPELINE INTEGRITY

## 01 THE MECHANISM

**Software Bill of Materials**  
Cataloging every component, library, and dependency that makes up a **shipped application**.

**Dependency Vulnerability Scanning**  
Checking every component against **known vulnerability databases** before and after it ships.

**Build & Signing Integrity**  
Verifying that code was built from trusted sources and **hasn't been tampered with in transit**.

## 02 THE FLOW

**The Build Cycle** CONTINUOUS

1

Developers pull dependencies from **public and internal registries** into the codebase.

2

Automated scanning checks each dependency's SBOM against **vulnerability and license databases**.

3

Signed, verified artifacts move through the pipeline into **a build that can be traced back to its source**.

**The Compromise Cycle** LIVE EVENT

1

An attacker compromises a widely-used dependency, build tool, or **maintainer account upstream**.

2

The tainted component flows downstream into every application that pulls it, **often silently**.

3

SBOM records and signing verification are what let defenders **identify and contain exposure fast**.

## 03 CORE PILLARS & COVERAGE

| PILLAR                     | WHAT IT COVERS                                                                       |
|----------------------------|--------------------------------------------------------------------------------------|
| Software Bill of Materials | Maintains a complete, current inventory of every component in a shipped application. |
| Dependency Scanning        | Continuously checks components against known vulnerability and license databases.    |
| Code Signing               | Cryptographically verifies that code came from a trusted source and wasn't altered.  |
| Pipeline Integrity         | Locks down build systems so only verified, authorized code reaches production.       |

## 04 SUPPLY CHAIN APPROACHES COMPARED

| SIGNAL                                    | SUFFICIENT ALONE? | WHAT IT MISSES                                                        |
|-------------------------------------------|-------------------|-----------------------------------------------------------------------|
| Trusting Vendor Security Claims           | No                | Claims aren't verification; you still don't know what's in the build. |
| Dependency Scanning Without an SBOM       | No                | Flags known issues but can't account for what isn't tracked at all.   |
| Manual Dependency Updates Only            | No                | Falls behind the pace new vulnerabilities are disclosed each day.     |
| Unsigned Internal Build Artifacts         | No                | Nothing stops a tampered build from reaching production undetected.   |
| <b>SBOM + Scanning + Signed Pipelines</b> | Yes               | Tracks, verifies, and secures every link from dependency to deploy.   |

SOFTWARE SUPPLY CHAIN.

TRACK · VERIFY · SECURE

05 PROGRAM OPERATIONS

**Most of the code is someone else's code.** The average application is mostly dependency code, not code written in-house.

**A single compromised package can cascade.** One tainted upstream library reaches every downstream project that depends on it.

**Unpatched dependencies age like unlocked doors.** Known vulnerabilities sit exploitable until someone updates the version.

**SBOMs only help if kept current.** A bill of materials from last release doesn't describe today's build.

06 CONTROL MAPPING

**NIST SP 800-218**

**Secure Software Development Framework** covering the full build and release lifecycle.

**EXECUTIVE ORDER 14028**

**Federal requirement** driving SBOM adoption across software vendors.

**SLSA FRAMEWORK**

**Supply-chain levels** grading build and provenance integrity.

**CISA SECURE BY DESIGN**

**Guidance pushing security ownership upstream** into development.

07 WHAT THIS DOES NOT SOLVE

**Zero-day vulnerabilities**

An SBOM shows what's there, not **what's about to be discovered**.

**Malicious code in updates**

A signed, expected-looking update can still **carry a hidden backdoor**.

**Runtime application behavior**

Supply chain controls secure what ships, not **what it does once running**.

**Compromised maintainer accounts**

A trusted maintainer's account can still **push malicious code that passes checks**.

08 MYTHS VS. REALITY

|                                                                                   |                                                                                                                         |
|-----------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------|
| <div>MYTH</div> <div>"Open source is less secure than commercial software."</div> | <div>REALITY</div> <div>Risk comes from <b>unmanaged dependencies</b>, not the license type.</div>                      |
| <div>MYTH</div> <div>"An SBOM alone protects the software supply chain."</div>    | <div>REALITY</div> <div>An SBOM is inventory, not defense — it needs <b>scanning and response</b> paired with it.</div> |
| <div>MYTH</div> <div>"Scanning once at release is enough."</div>                  | <div>REALITY</div> <div>New vulnerabilities are <b>disclosed daily</b> against code already shipped.</div>              |
| <div>MYTH</div> <div>"Only direct dependencies matter."</div>                     | <div>REALITY</div> <div>Transitive dependencies carry <b>just as much risk</b> as direct ones.</div>                    |
| <div>MYTH</div> <div>"Code signing guarantees the code is safe."</div>            | <div>REALITY</div> <div>Signing verifies origin and integrity, <b>not the absence of vulnerabilities</b>.</div>         |

09 SPEAK THE PROGRAM — KEY TERMS

**SBOM**

Software Bill of Materials: a complete inventory of **components in a piece of software**.

**TRANSITIVE DEPENDENCY**

A dependency pulled in indirectly by **another dependency**, not declared directly.

**CODE SIGNING**

Cryptographic verification that code came from a **trusted, unaltered source**.

**SLSA**

Supply-chain Levels for Software Artifacts: a framework grading **build provenance integrity**.

**PROVENANCE**

Verifiable metadata describing **how and where an artifact was built**.

**TYPOSQUATTING**

Publishing malicious packages under names that **mimic popular legitimate ones**.