

SOFTWARE SUPPLY CHAIN SECURITY.

Most of what ships in modern software was never directly chosen by anyone who built it — **the supply chain attack surface lives in the dependencies of dependencies, and in the build pipeline that assembles them, not just the code a team actually wrote.**

- SBOM & DEPENDENCY INVENTORY
- DEPENDENCY RESOLUTION HARDENING
- SIGNED BUILD PROVENANCE
- MAINTAINER TRUST REVIEW

01 THE MECHANISM



Dependency Confusion

A build system configured to check public registries alongside private ones can be tricked into pulling an attacker's package instead of an internal one, just by publishing a same-named package with a higher version number — **the resolver does exactly what it was designed to do, against the wrong source.**



Build Pipeline Tampering

Compromising the build process itself, rather than the source repository, lets an attacker inject malicious code into an artifact that still traces back to a clean, reviewed codebase — **the audit that would have caught it was checking the wrong stage of the pipeline.**



Transitive Dependency Exposure

A typical application's own code is a small fraction of what actually ships — most of the attack surface comes from dependencies of dependencies, layers removed from anything a developer chose. **Nobody consciously decided to trust most of what's running.**

02 THE FLOW

The Verified Software Supply Chain

CONTAINED

1

An SBOM is generated for every build, giving a complete, machine-readable inventory of every direct and transitive dependency that shipped.

2

Internal package names are explicitly scoped and pinned ahead of public registries, so **a same-named public package can never silently outrank the legitimate internal one.**

3

Every artifact carries cryptographic provenance tying it to a specific source commit and build process, so **a tampered build fails verification before it's ever deployed.**

The Unverified Software Supply Chain

EXPOSED

1

No SBOM exists, so nobody has a real inventory of what's actually shipping inside the application, direct or transitive.

2

Public and private package registries are checked with no strict priority or namespace scoping, so **a malicious public package with a higher version number silently replaces the trusted internal one.**

3

Build artifacts carry no provenance or signature verification, so **a compromised build pipeline can inject code that still appears to trace back to a clean source repository.**

03 CORE PILLARS & COVERAGE

PILLAR	WHAT IT COVERS
Software Bill of Materials (SBOM)	Generates a complete, machine-readable inventory of every direct and transitive dependency in a build, making otherwise invisible supply chain risk visible and auditable.
Dependency Resolution Hardening	Scopes and pins internal package namespaces explicitly ahead of public registries, closing the dependency confusion path that lets a same-named public package silently win.
Build Provenance & Artifact Signing	Cryptographically ties every artifact to the specific source commit and build process that produced it, so tampering anywhere in the pipeline breaks verification.
Maintainer & Registry Trust Review	Evaluates the maintenance health and account security of critical dependencies, since a single compromised or socially-engineered maintainer account can compromise everything downstream.

04 DETECTION APPROACHES COMPARED

APPROACH	SUFFICIENT ALONE?	WHAT IT MISSES
Source Code Review Alone	No	Audits the code in the repository, but a build-pipeline compromise injects malicious code after that review has already happened.
No SBOM / No Dependency Inventory	No	Leaves an organization unable to even identify whether a newly disclosed vulnerability affects anything it actually ships.
Unscoped Public/Private Package Resolution	No	Allows a same-named public package to silently outrank a legitimate internal one whenever it claims a higher version number.
Unsigned, Unverified Build Artifacts	No	Provides no way to confirm that what's deployed actually matches what the source and build process were supposed to produce.
SBOM + Resolution Hardening + Signed Provenance	Yes	Covers inventory, resolution integrity, build verification, and upstream trust together, rather than any one alone.

SOFTWARE SUPPLY CHAIN SECURITY.

INVENTORY · HARDEN · VERIFY

05 PROGRAM OPERATIONS

Most of what ships isn't code an organization wrote or chose directly. Transitive dependencies, layers removed from any direct decision, make up most of a typical application's actual attack surface.

The build pipeline is a target precisely because source review doesn't cover it. SolarWinds and the XZ Utils backdoor both compromised the build process, evading exactly the review most organizations rely on.

An SBOM only has value if it's actually queried. Generating an inventory and never checking it against new vulnerability disclosures provides the appearance of visibility without the substance of it.

Open-source maintenance is often a single point of failure disguised as free infrastructure. A handful of unpaid volunteers maintaining widely-depended-upon packages is a structural risk on its own.

06 CONTROL MAPPING

EXECUTIVE ORDER 14028

The 2021 U.S. federal directive requiring an SBOM for software sold to the federal government, driving broader SBOM adoption industry-wide.

SLSA (SUPPLY-CHAIN LEVELS FOR SOFTWARE ARTIFACTS)

A maturity framework, originating at Google and now under the Linux Foundation, defining escalating build-integrity guarantees against build-pipeline tampering.

NIST SP 800-218 (SSDF)

The Secure Software Development Framework, defining practices for building, verifying, and maintaining software supply chain integrity.

CYCLONEDX / SPDX

The two dominant SBOM formats, providing a standardized, machine-readable way to represent a software component inventory.

07 WHAT THIS DOES NOT SOLVE

First-Party Code Vulnerabilities

An accurate dependency inventory doesn't fix a **flaw in the code an organization wrote itself**, which needs its own review process.

Zero-Days in Trusted Dependencies

Even a fully signed, verified dependency can carry a **vulnerability nobody has discovered yet**, regardless of supply chain trust.

Runtime Application Behavior

Verifying what shipped doesn't monitor **what the application does once it's running**, a separate runtime-security concern.

Insider Threats With Legitimate Access

Provenance and signing verify the process, but don't stop an **insider introducing malicious changes through legitimate channels**.

08 MYTHS VS. REALITY

MYTH "If the source code repository looks clean, the shipped software is clean."	REALITY SolarWinds and the XZ Utils backdoor both showed that compromising the build pipeline itself can inject malicious code that never appears in the reviewed source.
MYTH "An SBOM is mainly a compliance checkbox."	REALITY An SBOM is the only practical way to answer "does this newly disclosed vulnerability affect anything we ship?" across every transitive dependency, not just direct ones.
MYTH "Dependency confusion only affects careless organizations."	REALITY It exploits a default package-resolution behavior present in many build systems out of the box, not just an obvious misconfiguration.
MYTH "A popular open-source package is safe because so many people use it."	REALITY Popularity doesn't guarantee maintenance capacity or account security — the XZ Utils backdoor targeted a project depended on by countless systems, maintained by very few people.
MYTH "Code signing alone proves an artifact is trustworthy."	REALITY Signing only proves the artifact wasn't altered after signing — it doesn't prove the build process that produced it wasn't itself compromised, which is exactly what provenance attestation is for.

09 SPEAK THE PROGRAM — KEY TERMS

SBOM (SOFTWARE BILL OF MATERIALS)

A **complete, machine-readable inventory** of every component and dependency, direct and transitive, in a piece of software.

DEPENDENCY CONFUSION

An attack that tricks a build system into pulling a **malicious public package instead of a legitimate internal one** of the same name, via a higher declared version number.

SLSA (SUPPLY-CHAIN LEVELS FOR SOFTWARE ARTIFACTS)

A framework defining **escalating levels of build integrity and provenance** guarantees, designed to prevent build-pipeline tampering.

BUILD PROVENANCE

A verifiable, cryptographically signed record of **exactly which source and build process produced a given artifact**.

TYPOSQUATTING

Publishing a malicious package under a name that's a **near-identical misspelling of a popular legitimate package**, hoping developers mistype an import.

TRANSITIVE DEPENDENCY

A dependency pulled in indirectly, **several layers removed from anything a developer chose directly**, often unknown to the team shipping the final product.