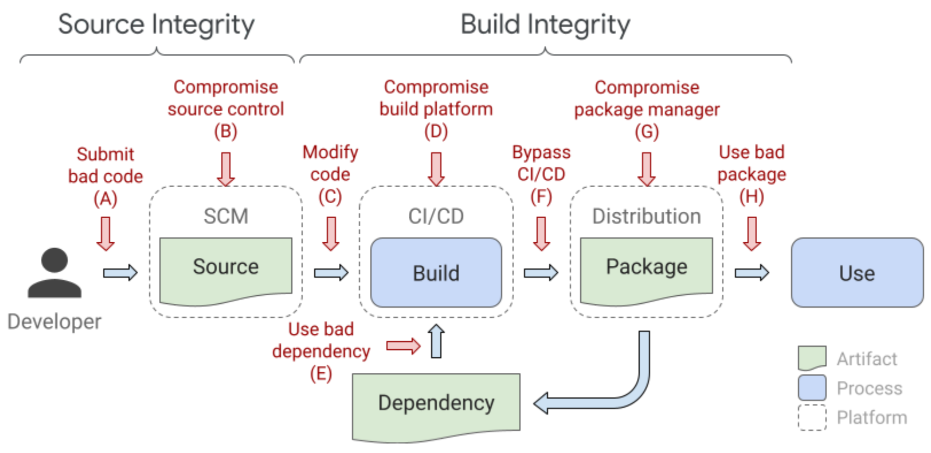
Software Supply Chain Integrity

Submitter: *Eric Brewer, on behalf of Google, LLC*
Topic: *(5) Guidelines for software integrity chains and provenance*
Speakers: *Mark Lodato, Staff Software Engineer, lodato@google.com*

The recent high-profile supply chain attacks like those on SolarWinds and CodeCov were integrity attacks—unauthorized modifications to software packages. These attacks appear to be on the rise, and worse still, supply-chain vulnerabilities are widespread and without comprehensive solutions.

We propose Supply chain Levels for Software Artifacts (SLSA, pronounced "salsa") as an end-to-end solution, based on Google's internal "Binary Authorization for Borg" that has been in use for the past 8+ years and that is mandatory for all of Google's production workloads. We envision SLSA to work in tandem with existing solutions such as SBOMs to address a broad range of threats in a comprehensive, easily adopted framework for the wider industry.

Supply-chain integrity is a multifaceted problem. Threats are present at each point in the software supply chain, as pictured in the following diagram. For example, the SolarWinds attack involved a compromised build platform (D), while the CodeCov attack involved compromised package uploader credentials (F).



We broadly divide integrity into two main areas:

Source Integrity and Availability: Ensure that changes to the source code reflect the intent of the organization that owns the source without any tampering, and that the code and change history remain available for investigations and incident response.

Build Integrity: Ensure that packages are built from the correct, unmodified sources and dependencies according to the build recipe defined by the software producer, and that artifacts are not modified as they pass between development stages.

SLSA addresses all of the integrity threats (A) through (H) with emphasis on the most pressing threats. SLSA is a framework for ensuring that software packages meet well-defined integrity standards, such as code review, change history, build isolation, and provenance generation.

Automated policy enforcement gives consumers confidence in the software's integrity, and simple numeric "levels" enable incremental adoption and succinctly convey security posture.

Provenance and policies

We believe that trustworthy software metadata is essential to prevent and detect attacks on integrity and for incident response. SLSA recommends in-toto attestations, and particularly provenance[1], to provide metadata that: 1) is authenticated; 2) contains the necessary immutable[2] information, such as dependency trees, to address supply chain threats; 3) is generated easily and automatically; and 4) is interoperable across platforms and build steps. Importantly, these attestations can be assembled into a directed acyclic graph to analyze the supply chain through transitive dependencies back to original source code.

A policy engine consumes this metadata as evidence that the software was built according to the expected process, and that this process met standard integrity guarantees. For example, a policy might require a package to be built from a particular git repository by a particular build service; the engine would reject packages that lack the expected build metadata. Each policy has a numeric level (1-3) describing its security strength, which consumers can rely on to quickly understand security posture.

Trusting platforms, not users

Our goal is to minimize the number of parties that consumers of software must trust, and to have controls to prevent abuse of that trust. SLSA encourages a small number of high-integrity, trustworthy source, build, and packaging platforms because they are hard to build correctly. SLSA requires these platforms to implement strong administrative access controls such as multi-party authorization, mandatory audit mechanisms, and securely built and deployed software. Going forward, we can reduce the need to trust any single platform through techniques such as reproducible builds and/or trusted execution environments.

Importantly, we place responsibility for producing provenance with the build platform and not with user-provided tooling that runs on top of the platform. This reduces the scope of trusted actors and mitigates risks from project members, such as abuse of provenance signing keys.

Integration with SBOM

SBOM is important for supply-chain security, particularly vulnerability management and IP licensing, but it is not necessarily the same as SLSA's provenance. SLSA's provenance is essentially an easy-to-generate "intermediate form" that captures the integrity aspects of the supply chain. It can be used to produce an SBOM on demand, possibly with supplemental information whose integrity is also protected by SLSA provenance. This has an advantage over traditional SBOM in that it provides authentication and integrity, stronger guarantees about completeness, and practical traversal of the dependency graph to arbitrary depth.

In summary, SLSA is a comprehensive end-to-end framework to address integrity threats. It is practical to deploy, complementary to SBOM, and based on a model proven to work at scale.

---

[1] We use the term "provenance" more broadly than SBOM, to mean how an artifact was generated.
[2] Artifacts are immutable because they are "sealed" by signatures and hashes, ensuring tamper resistance and preserving contents. The tree is immutable as all of the nodes and links are immutable.