Make Your CI More Secure And Less Spicy With Some SLSA

OWASP NZ DAY 2024
6 SEPTEMBER 2024
BY JAMES COOPER

Thank You to Our Sponsors and Hosts!

BASTION

SECURITY GROUP

DATACOM

plexure

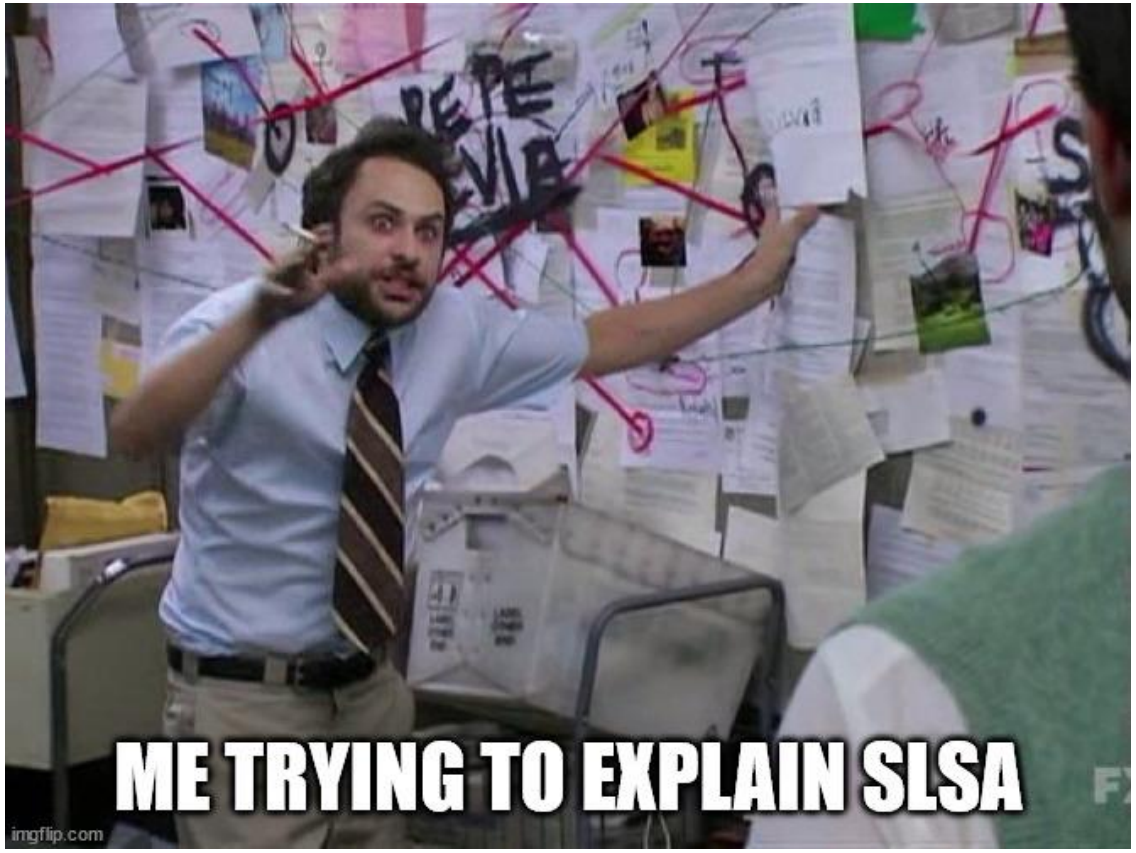
Without them, this Conference couldn't happen.

Who's this guy?

- Security-interested software developer (these days)
- Application Developer at 2degrees 
- Ph.D. in Computer Science from University of Auckland
- Still getting my own head around SLSA (& friends)
- Assume (for lack of time) that you already know about supply-chain security issues

Overview

So, SLSA, eh?



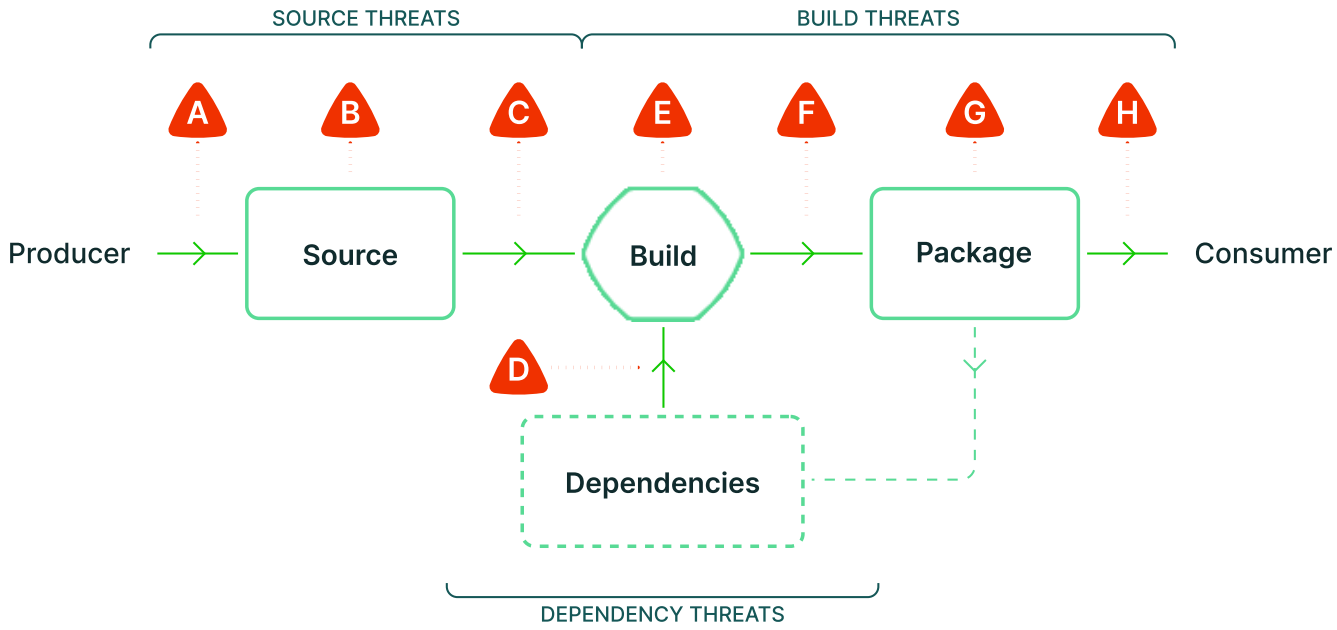
SLSA ‘from 10,000 feet’

- “Supply-chain Levels for Software Artifacts”
  - Pronounced like “salsa”
- Official website at https://slsa.dev/
  - *“It’s a security framework, a checklist of standards and controls to prevent tampering, improve integrity, and secure packages and infrastructure. It’s how you get from "safe enough" to being as resilient as possible, at any link in the chain.”* (taken from the website’s frontpage)
- Aims to make software supply chain attacks much harder to achieve
- Originally from Google, now an Open Source Security Foundation (OpenSSF) project
- Released v1.0 in April 2023, but still work-in-progress
  - Initial release scaled back original ambitions to get it out the door

SLSA & Friends

- SLSA is part of a broader (still nascent) interconnected system
  - SLSA
  - FRSCA
  - SigStore
  - Tekton
  - In-toto attestation
  - Etc…
- A rather tangled web, if you get far enough into it
  - Everything seems to be (semi-)independent, too
- Will just focus on (part of) SLSA today, though, since we have ≤ 30 minutes, not 3 days
  - ∴ discussion at high level only ☹

What does SLSA currently *target*?



SOURCE THREATS

- ✗ A Submit unauthorized change
- ✗ B Compromise source repo
- ✓ C Build from modified source

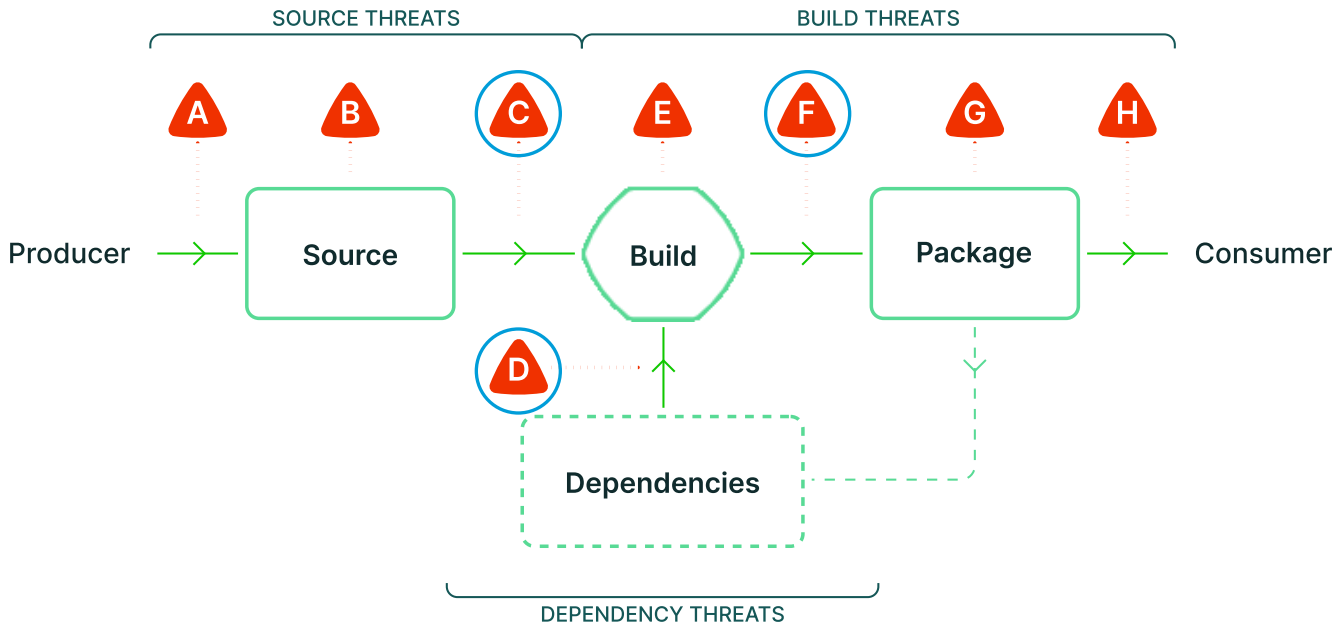
DEPENDENCY THREATS

- ✓ D Use compromised dependency

BUILD THREATS

- E Compromise build process ✓
- F Upload modified package ✓
- G Compromise package registry ✗
- H Use compromised package ✗

What does this talk target?



SOURCE THREATS

- ✗ A Submit unauthorized change
- ✗ B Compromise source repo
- ✓ C Build from modified source

DEPENDENCY THREATS

- ✓ D Use compromised dependency

BUILD THREATS

- E Compromise build process ✗
- F Upload modified package ✓
- G Compromise package registry ✗
- H Use compromised package ✗

Elements of SLSA

SLSA Terminology

- Build
  - *"Process that transforms a set of input artifacts into a set of output artifacts. The inputs may be sources, dependencies, or ephemeral build outputs."*
- Artifact
  - *"An immutable blob of data; primarily refers to software, but […] can be […] any artifact."*
- Dependency
  - *"Artifact that is an input to a build process but that is not a source. […] it is always a package."*
- Source
  - *"Artifact that was directly authored or reviewed by persons, without modification."*
- Package
  - *"Artifact that is 'published' for use by others."*

More SLSA Chunks

- As of v1.0, SLSA has one ‘track’, the Build Track, with two main foci
  - Producing ‘provenance’ for built artifacts
  - Using provenance to verify artifacts consumed as dependencies

- Four levels of security assurance (higher levels depend on lower ones)
  - Level 0: No Guarantees
  - Level 1: Provenance Exists
  - Level 2: Hosted Build Platform
  - Level 3: Hardened Builds

- Some levels currently only achievable with certain build platforms

Provenance

- "Attestation (metadata) describing how the outputs were produced, including identification of the platform and external parameters."
- Required for SLSA Build Level 1+
- At higher levels, provenance is generated and signed by the build platform
- Producers produce it to accompany their packages
- Consumers use it to verify consumed packages match expectations

SLSA Provenance ≠ SBOMs

- Sounds an awful lot like a Software Bills of Materials (SBOMs)?

- SLSA Provenance & SBOMs are related, but not the same
  - SBOM describes what went into an artifact
  - Provenance covers how it was made

- SLSA project likens it to food production
  - SBOM lists ingredients
  - Provenance describes food safety standards followed by manufacturers

- Use both!

SLSA in your CI

Differentiated Integration

- Different languages and ecosystems have different tooling available so far
  - Only some have much support at all…
  - Some generic tools also
- Different build systems have different maximum SLSA Levels
  - Currently, only Google Cloud and GitHub Actions are rated up to L3
  - Discuss GitHub Actions today—widely used & available
- Two main things you can do
  - Produce provenance
  - Verify provenance

Producing Provenance

- Used by consumers of your package to confirm that it's the genuine article

- The provenance describes how the package was built, including
  - Build platform
  - Build process
  - Build inputs

- Doesn't by itself do much to help producers on a **technical** level
  - Proper implementation suggests relatively secure CI, though
  - Can be beneficial on a reputational/social level—reflects well on you
  - Might (hopefully will) be required by customers in future

- Main focus of current security levels

Getting to Level 1

- Level 0 is a lack of any assurance
  - E.g., developer doing local build
  - Generally, no intention to provide any form of assurance

- Level 1 means
  - Build processes are consistent
  - ‘Some’ provenance is produced and provided for a package
  - No requirement for cryptographically signing provenance

- Must meet level 1 to go further

Getting to Levels 2 & 3

- Level 2
  - Builds are always performed on dedicated infrastructure
  - Provenance is digitally signed
  - Difficult for external parties to fake your packages
    - Still potentially vulnerable to insider threats
- Level 3
  - Build systems must be hardened against tampering
  - Build platform ensures all builds are isolated
  - Key material for signing provenance must never be exposed
  - Makes it extremely difficult for anyone to muck with build process
    - Does little about malicious updates to genuine sources

Producing Provenance with GHA

- GitHub has published a basic provenance action, `actions/attest-build-provenance`
  - Run it on an artifact produced by your Actions workflow to create signed provenance
- The SLSA project also has builders and generators
- Builders both run the build process and produce provenance
  - Builders for NPM, Go, Java & Docker containers—meet L3 requirements
- Generators just produce provenance for other artifacts
  - Generators available for generic artifacts and container images
  - Generators pre-date GitHub's Actions
- Unclear whether GitHub's action and the generators meet L2 or L3 requirements

Consuming Provenance

- Details how and where a package was built
- Provenance is digitally signed, at higher levels
- Verifying provenance signature confirms a package came from the real producers
  - Stops sneaky swaps of packages for illegitimate knock-offs
- Doesn’t by itself stop malicious corruption of legitimate builds
  - Out-of-scope of SLSA

Consuming Provenance with GHA

- Verify artifacts produced by GitHub Action via GitHub's CLI
  - `gh attestation verify …`
  - Not totally clear whether it works for other provenance generators
- SLSA produce their own CLI to verify artifacts produced by their builders and generators
  - Also supply an installer action to use it in GitHub Actions
- Others out there, but make either of these your first choice
- In all cases, result can be as simple as a binary 'yes/no verified'
  - Stop your CI process on a 'no'!

DIY sans SLSA

No Silver Bullets

- SLSA doesn't address *all* possible issues
  - Left-pad, anyone?
  - Polyfill.io
  - So-called 'protestware'

- Many issues can be prevented without SLSA, anyway

- Do 'The Fundamentals' well
  - A lot of what SLSA is telling us, when you get down to it
  - Can achieve much of the same benefits without SLSA! (DIY)

‘The Fundamentals’?

- Automate the heck out of everything possible
- Use separate, dedicated build infrastructure
- Delegate responsibility to trusted platforms, where appropriate
- Closely control privileges and access permissions (and monitor)
- Cryptographically sign outputs/artefacts
- Verify cryptographic signatures, and file & git commit hashes, too

The End

Some potentially useful links:

- https://slsa.dev/
- https://openssf.org/
- https://github.com/slsa-framework/slsa-github-generator
- https://github.com/slsa-framework/slsa-verifier
- https://github.com/actions/attest-build-provenance
- https://docs.github.com/en/actions/security-for-github-actions/using-artifact-attestations/using-artifact-attestations-to-establish-provenance-for-builds
- https://docs.docker.com/build/metadata/attestations/

Examples of supply chain attacks

- Webmin
  - Someone got into the build server and updated a local copy of a source file
  - Build server apparently didn't use source directly from version control
- Event-stream
  - Updated NPM package to (temporarily) add a new dependency
  - Version of dependency published to NPM differed from source on GitHub
  - Added extra code to target certain Bitcoin wallets
- CodeCov
  - Gained access to CodeCov's cloud environment
  - Replaced official script asset with malicious version that users accessed instead
- The sorts of things SLSA aims to prevent

SLSA vs The Examples

Examples Revisited

- Webmin
  - File changed on build server
  - Treat source as artifact, verify provenance before building
  - Hardened build system could help, too
- Event-stream
  - Added bit to NPM package to target Bitcoin wallets
  - Recursively applying provenance verification to dependencies would have detected modification
- Codecov
  - Swapped cloud-hosted file for malicious version
  - Verifying the package would have shown it wasn't built the proper way