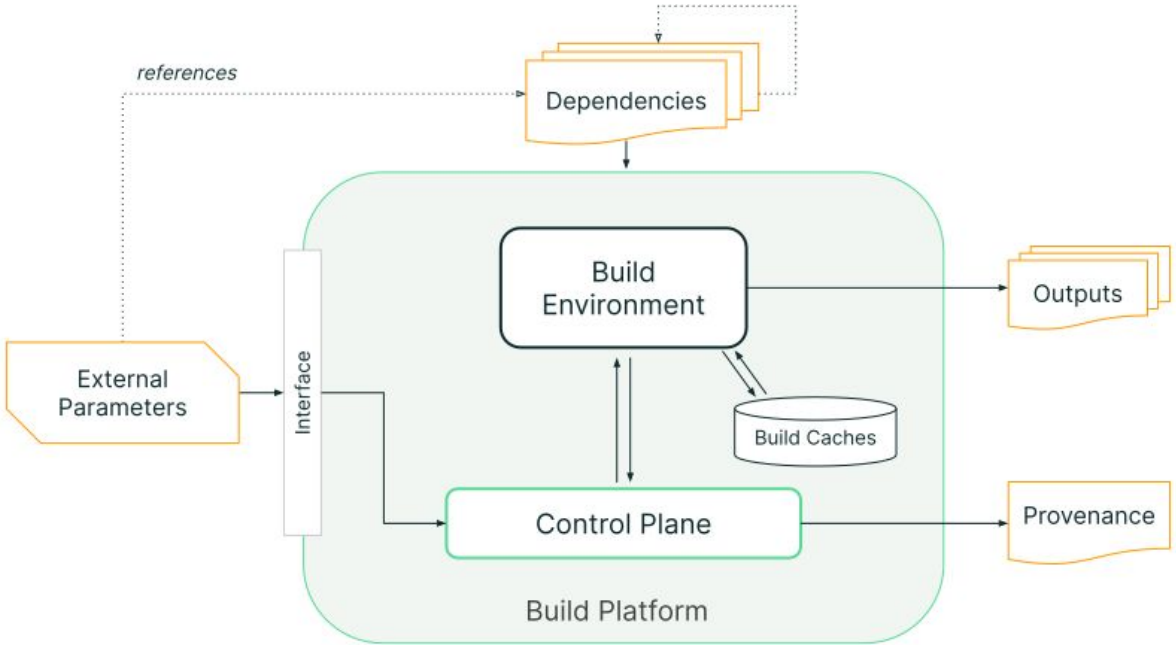
Securing build platforms

Linux Plumbers Conference 2023
Build Micro Conference

Joshua Lock
Verizon

Build platform?

System that allows tenants to run builds. Technically, it is the **transitive closure of software and services that must be trusted** to faithfully execute the build. It **includes software, hardware, people, and organizations.**



Motivation

Open source is built on trust

---

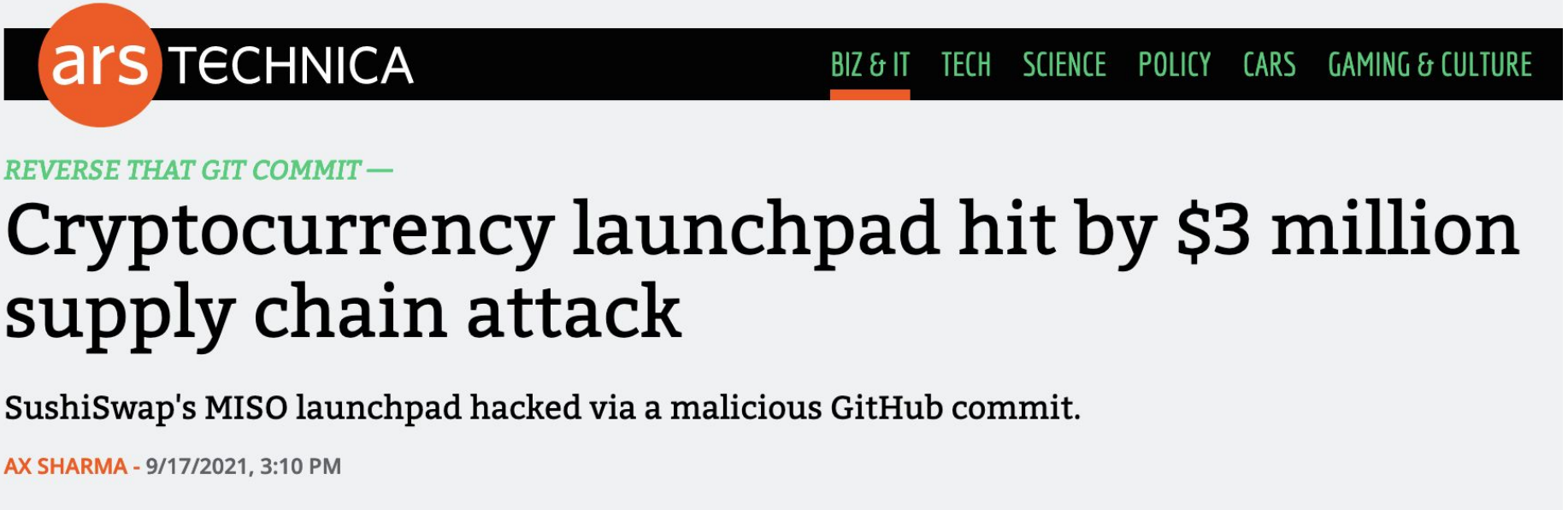
source + recipe + builder = artefact

⇒ artefact produced from expected/canonical source repository

⇒ artefact produced using expected recipe

⇒ artefact produced on expected/trusted builder

but, attackers are inside the network...



Changes to Git commit workflow

From:	Nikita Popov	**Date:**	Sun, 28 Mar 2021 22:52:24 +0000
Subject:	Changes to Git commit workflow		
Groups:	php.doc php.internals		

```
Hi everyone,

Yesterday (2021-03-28) two malicious commits were pushed to the php-src
repo [1] from the names of Rasmus Lerdorf and myself. We don't yet know how
exactly this happened, but everything points towards a compromise of the
git.php.net server (rather than a compromise of an individual git account).
```

- Saving a malicious HTML attachment could trigger and XSS vulnerability.

Webmin 1.882 to 1.921

Remote Command Execution [CVE-2019-15231]

- Webmin releases between these versions contain a vulnerability that allows remote command execution! Version 1.890 is vulnerable in a default install and should be upgraded immediately - other versions are only vulnerable if changing of expired passwords is enabled, which is not the case by default.
  Either way, upgrading to version 1.930 is strongly recommended. Alternately, if running versions 1.900 to 1.920, edit `/etc/webmin/miniserv.conf`, remove the `passwd_mode=` line, then run `/etc/webmin/restart` command.
  ▶ **More details..**

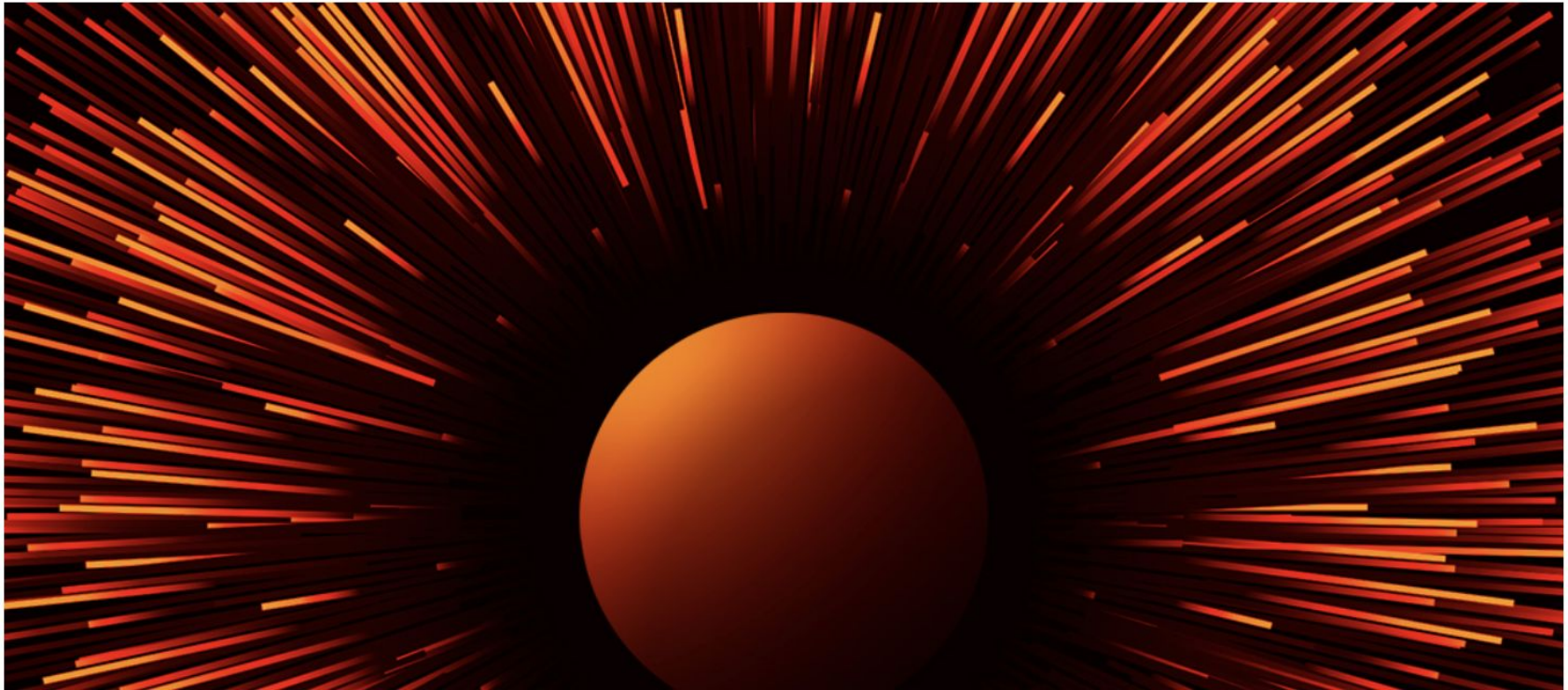
event-stream vulnerability explained

November 27, 2018 • 11 min read

If you work with JavaScript at all, you probably saw a ton of noise yesterday about a vulnerability in the event-stream npm package. Unfortunately, the actual forensic analysis of the issue is buried under 600+ comments on the GitHub issue, most of which are just people flaming about the state of npm, open source, etc. I thought that was a shame, because the vulnerability was actually exceptionally clever and technically interesting, and teaches some important lessons about maintaining security in JavaScript applications. So I decided to write an explainer detailing what happened, how the attack worked, and how the JavaScript community can better defend against similar attacks in the future.

SUNSPOT: An Implant in the Build Process

January 11, 2021 CrowdStrike Intelligence Team Counter Adversary Operations



Post-Mortem / Root Cause Analysis (April 2021)

Summary

On April 1, 2021, the Codecov team was alerted to a security event involving our Bash Uploader. The threat actor specifically targeted the Codecov Bash Uploader and used it to deliver a malicious payload to all Codecov users utilizing the Bash Uploader, The Codecov GitHub Action, The Codecov CircleCI Orb, and the Codecov Bitrise Step (collectively, the “Bash Uploaders”).

What to do?

Prior art

Binary Authorization for Borg

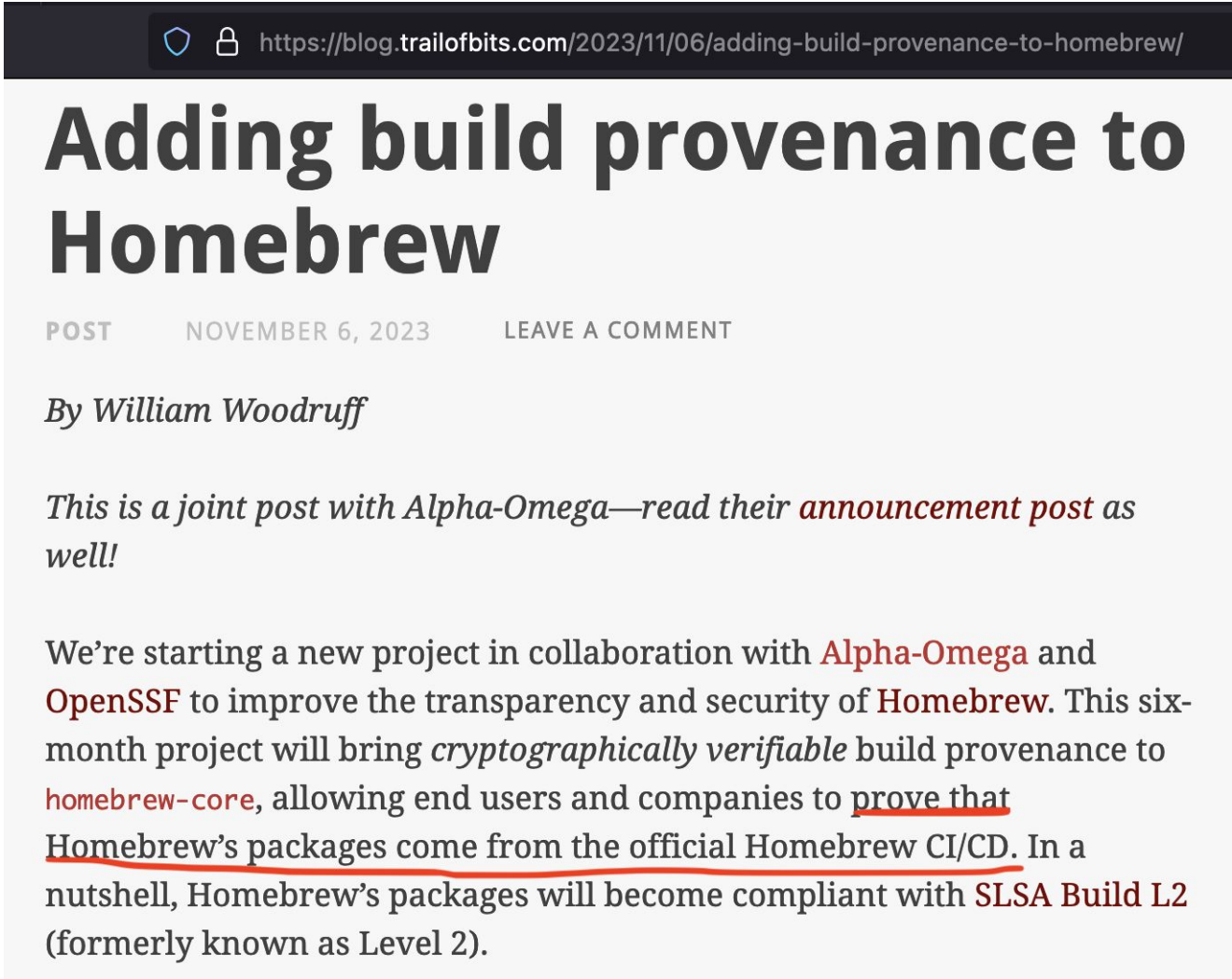
Send feedback

This content was last updated in September 2023, and represents the status quo as of the time it was written. Google's security policies and systems may change going forward, as we continually improve protection for our customers.

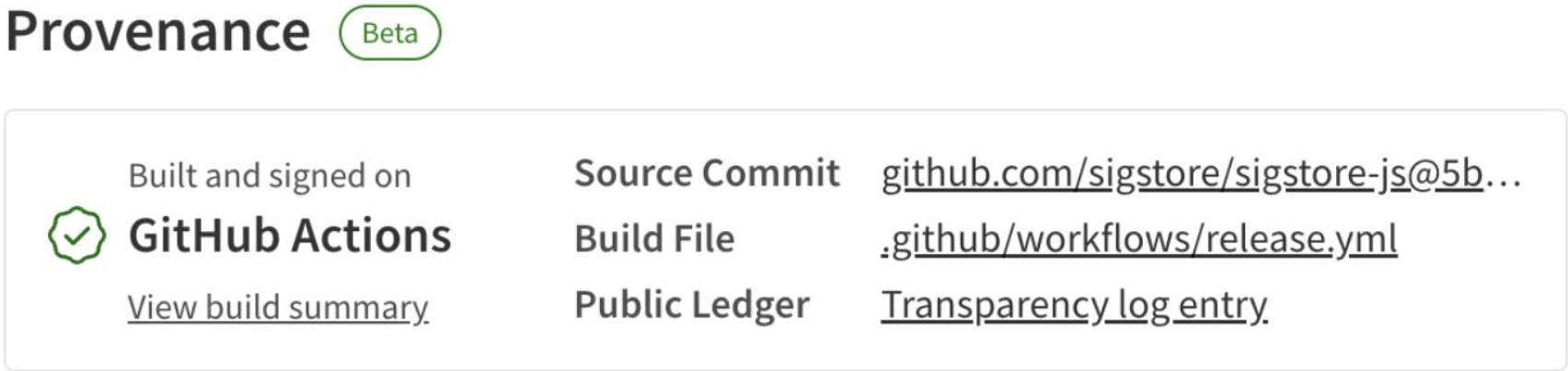
This document describes how we use code reviews, security infrastructure, and an enforcement check called *Binary Authorization for Borg (BAB)* to help protect Google's software supply chain against insider risk. BAB helps reduce insider risk because it ensures that production software is reviewed and approved before it's deployed, particularly when our code can access sensitive data. Since the original publication of this document, we have included key concepts of BAB into an open specification called Supply chain Levels for Software Artifacts (SLSA).

This document is part of a series of technical papers that describes some projects that the Google security team have developed to help improve security, including BeyondCorp and BeyondProd. For an overview of our infrastructure's security, see the Google infrastructure security design overview.

Adjacent spaces



Starting today, when you build your npm projects on GitHub Actions, you can publish provenance alongside your package by including the `--provenance` flag. This provenance data gives consumers a verifiable way to link a package back to its source repository and the specific build instructions used to publish it (see example on npmjs.com).



Linux distributions too

1 Software supply chain security at SUSE

Securing our software supply chain is a top priority for SUSE to protect our customers from security risks, known and zero-day vulnerabilities. Ensuring that no threat actor can inject malicious code into our build service systems is attested by industry-leading security certifications. Our teams continually work to certify all SUSE products, and develop security solutions to offer our customers the highest level of trust and reliability.

A new industry standardization effort named SLSA (Supply chain Levels for Software Artifacts), started by Google and driven by several industry stakeholders, aims to protect the integrity of the software supply chain.

SLSA defines four levels of assurance, going from basic requirements at level 1 to strict rules and documentation requirements at level 4.

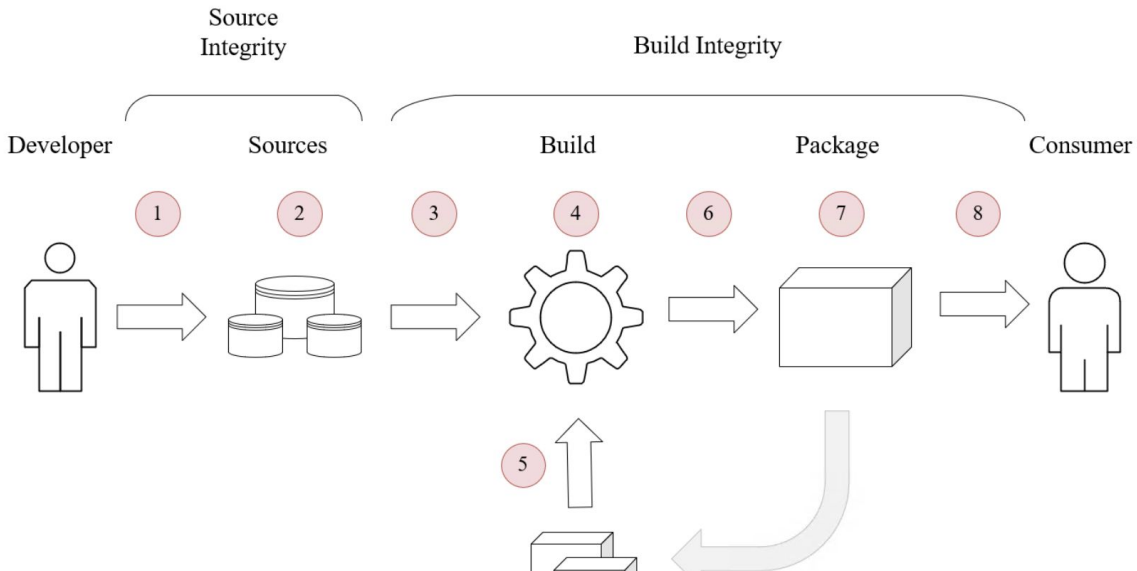
While the SLSA standard is still in development, SUSE already considers it as a great representation of needs for a secure product build environment, and we are adjusting our processes and tooling to meet the requirements of the highest assurance level 4.

Flatcar Container Linux Supply Chain Security and SLSA

The Supply Chain Levels for Software Artifacts (SLSA or 'salsa' for short) industry standard defines a checklist of standards and controls to prevent tampering, improve integrity, and secure packages and infrastructure in software projects. This document describes the Flatcar Container Linux project's current and planned compliance with the requirements of SLSA and provides a deep dive into the processes and mechanisms to secure the Flatcar project supply chain.

Our assessment is that Flatcar complies with SLSA Level 3. We are working to address the few remaining requirements for SLSA Level 4.

SLSA Threat model and requirements



SLSA
Supply-chain Levels for Software Artifacts

Generalise(BinAuthBorg) → SLSA

What is SLSA?

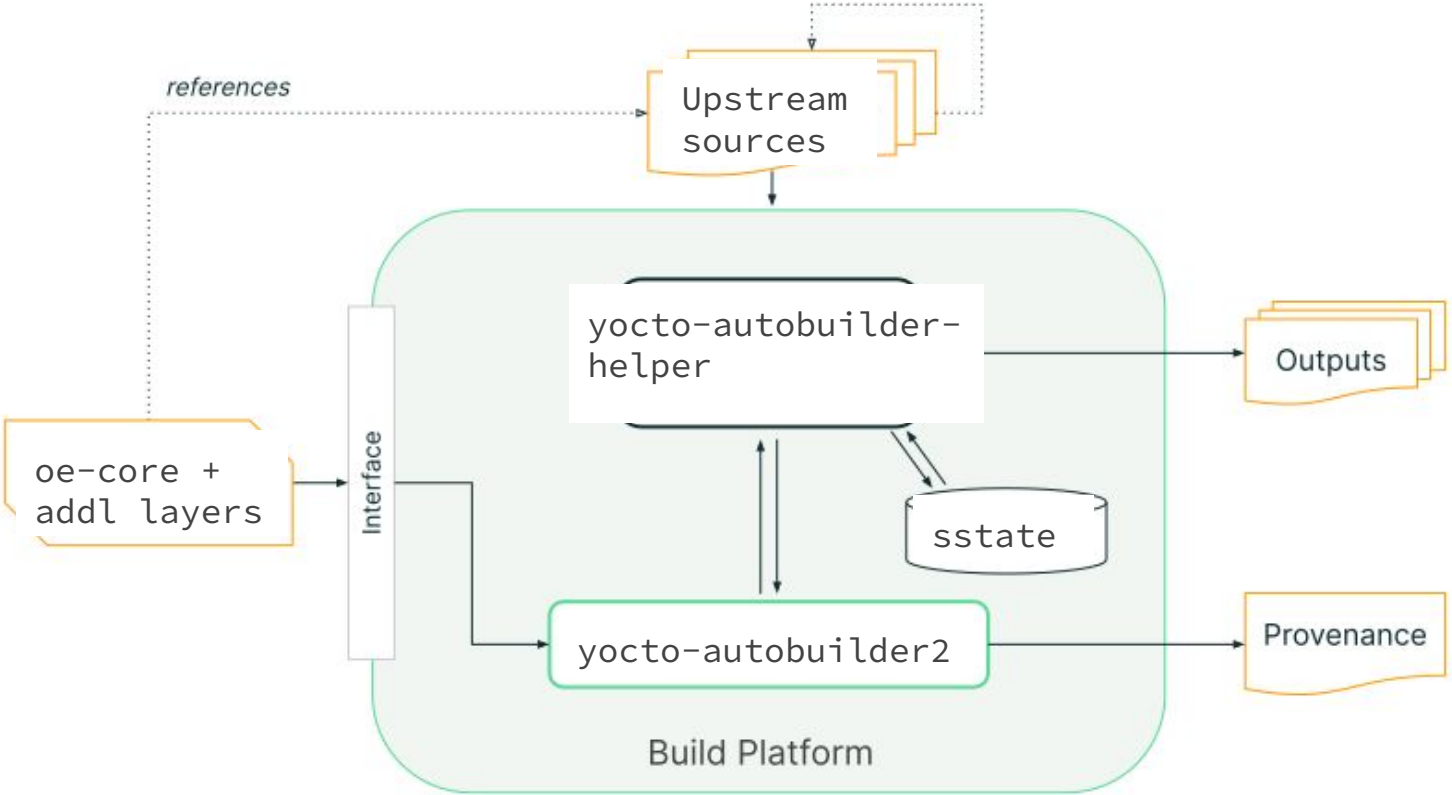
Supply-chain Levels for Software Artifacts, or SLSA ("salsa"). It's a security framework, a checklist of standards and controls to prevent tampering, improve integrity, and secure packages and infrastructure. It's how you get from "safe enough" to being as resilient as possible, at any link in the chain.

Key components of a SLSA architecture

Ecosystem adoption:

- Trusted platform(s)
  - Provenance generation
  - Isolation strength
- Expectations: TOFU, producer-defined, defined in source (i.e. Go), <your idea here>
- Verification: at registry/repository admission, at install/deploy, <your idea here>

Yocto Project build platform



SLSA for Yocto Project build platform

SLSA Build L1:

- Generate provenance as a buildbot plugin

SLSA Build L2:

- Sign generated provenance
- *Attacks like CodeCov and Webmin are detectable* 🎉

SLSA Build L3+

- TBD

References

- Binary authorization for Borg: https://cloud.google.com/docs/security/binary-authorization-for-borg
- Supply-chain Levels for Software Artifacts (SLSA): https://slsa.dev/
- npm provenance announcement: https://github.blog/2023-04-19-introducing-npm-package-provenance/
- Homebrew + SLSA: https://blog.trailofbits.com/2023/11/06/adding-build-provenance-to-homebrew/
- Software supply chain security @ SUSE: https://documentation.suse.com/sbp/server-linux/html/SBP-SLSA4/index.html
- Flatcar supply chain security + SLSA: https://www.flatcar.org/docs/latest/reference/supply-chain/

Keeping the conversation going

OpenSSF Securing Software Repositories WG (https://repos.openssf.org/)

- Build Provenance for All Package Registries: https://repos.openssf.org/build-provenance-for-all-package-registries
- Build Provenance and Code-signing for Homebrew: https://repos.openssf.org/proposals/build-provenance-and-code-signing-for-homebrew

OpenSSF Supply Chain Integrity WG (https://github.com/ossf/wg-supply-chain-integrity)