OWASP GLOBAL AppSec

VIENNA '26

JUN 25-26

25 years of open source security

FROM 0 TO SLSA LEVEL 3

A Practitioner's Field Guide

Mark Mishaev
Director of Engineering, Software Supply Chain Security · GitLab

What you'll learn

Whether you're starting your SLSA journey or stuck at Level 2, walk away with battle-tested patterns that work at scale.

- **Architecture** Provenance attestation architecture for CI/CD pipelines.
- **Integration** How to integrate SLSA verification without breaking existing workflows.
- **Real metrics** What SLSA costs in CI minutes — and what attacks it actually catches.
- **Pitfalls** Common implementation pitfalls and how to avoid them.

OWASP GLOBAL AppSec

VIENNA '26 JUN 25-26

25 years of open source security

WHY SLSA MOVED FROM A NICE SPEC TO A BUYER REQUIREMENT

PART 1 · WHY SLSA MATTERS NOW

SLSA moved from “nice spec” to buyer requirement

The buying conversation changed faster than most CI platforms changed their trust model.

Weekly headlines

XZ Utils, Ultralytics, and the Shai-Hulud npm worm reinforced one lesson: source review is not enough when the build chain is the attack path.

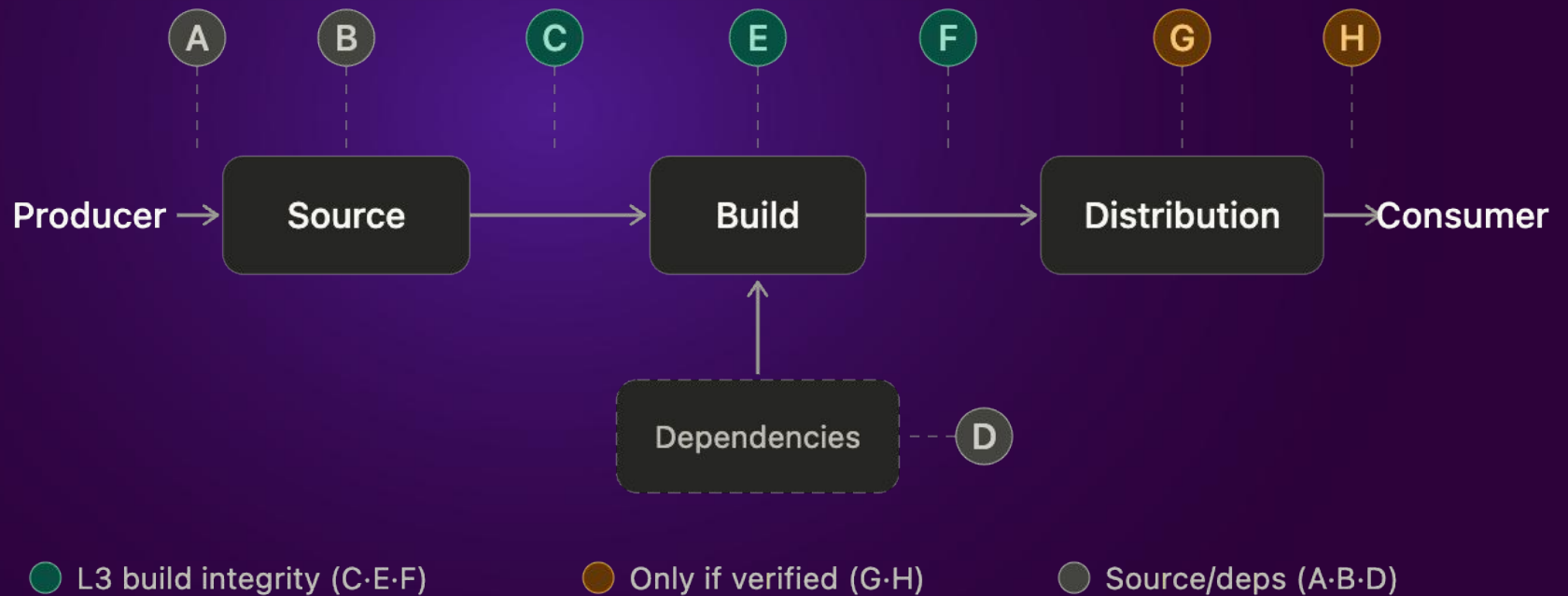
Regulatory pressure

The EU Cyber Resilience Act, EO 14028, and SSDF-style controls converge on one question: can you prove how the artifact was produced?

Customer demand

Enterprise buyers increasingly ask for provenance attestation in RFPs, and regulated customers revisit the gap at renewal.

The canonical SLSA supply-chain threat model



Most teams say "SLSA" when they mean Level 2

Level 3 is where the trust boundary moves: who signs, where they sign, and what infrastructure identity is provable.

L0

No provenance

You can build, but you cannot prove what happened.

L1

Provenance exists

Useful metadata, but easy to forge if the job itself is compromised.

L2

Platform-signed

Cryptographic evidence the build ran — usually enough for initial rollout and marketing claims.

L3

Isolated + unforgeable

Signing is outside the job, builds are truly isolated, and provenance cannot be faked by a privileged tenant.

The L2 plateau has three architectural blockers

The practical Level 3 test: if an attacker fully compromises a build job, can they still produce a trusted attestation?

1. In-job signing

If the build job can execute arbitrary code, it can sign arbitrary outputs. That breaks “unforgeable” immediately.

2. Shared runners

Container isolation is not platform isolation. Residual state, weak teardown, and co-located tenants all matter.

3. Weak trust roots

If verifiers cannot tie artifacts to platform records, runner identity, and a protected signing path, the provenance is decorative.

A trusted control plane outside the runner blast radius

Build output → platform-side verification → out-of-band signing → deploy-time verification.

Provenance generation Collect artifact digest, source ref, pipeline metadata, dependency evidence, and SBOM signals at the platform layer.

Signing infrastructure Use Sigstore-compatible signing and verification, but keep the signing decision in a service the build cannot influence.

Storage + verification Store in-toto attestations alongside OCI artifacts; make verification work in CI gates, CLI flows, and deploy controls.

Sigstore: keyless signing with Fulcio and Rekor

Short-lived, identity-bound certificates and a public log replace long-lived signing keys — nothing to steal, everything to audit.

Sigstore

The CNCF project for keyless signing. Sign with a short-lived identity instead of a stored key — Cosign is the client that drives it.

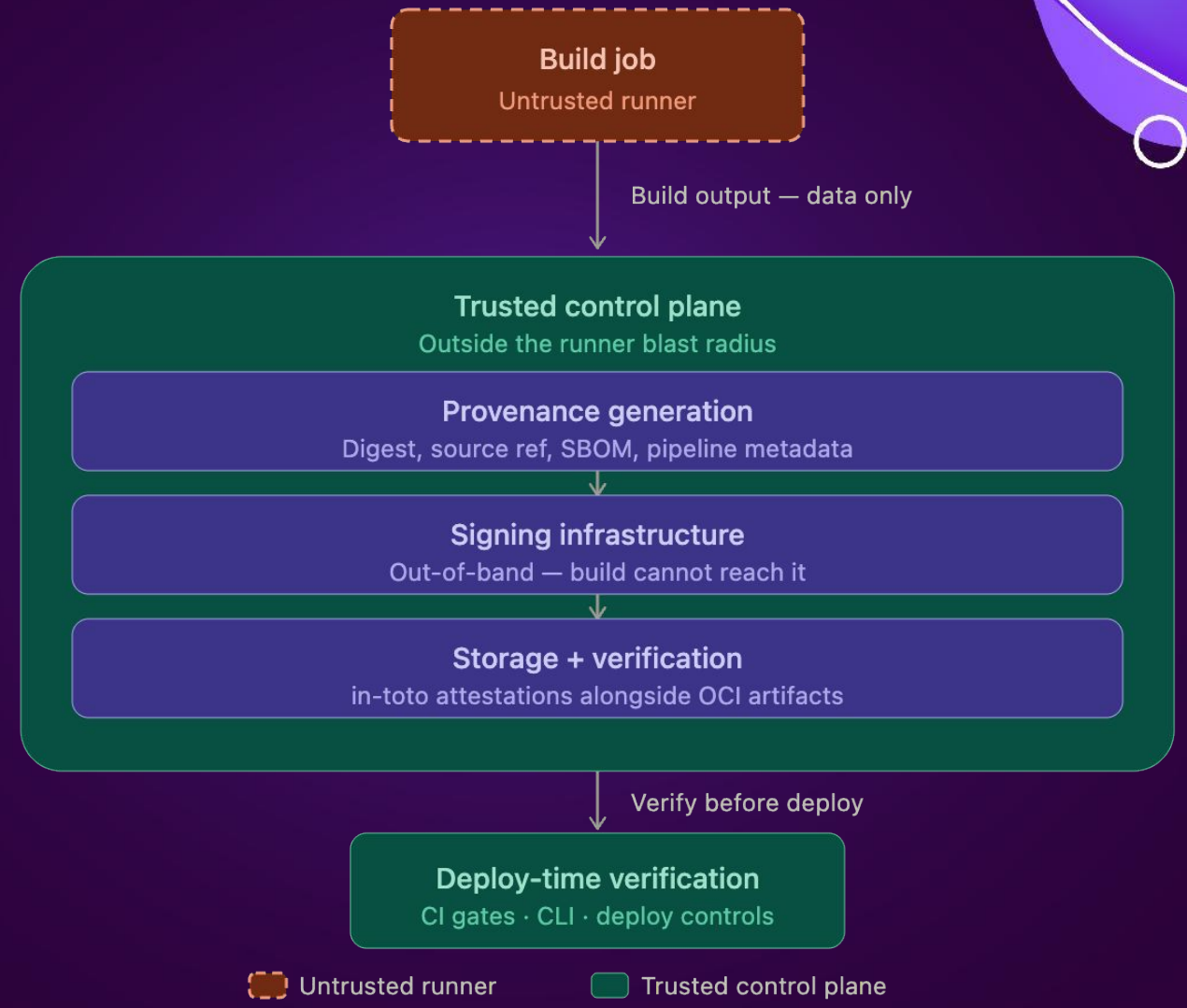
Fulcio

The certificate authority. Trades an OIDC identity for an ephemeral X.509 cert (~10 min) bound to that identity — the key never outlives the build.

Rekor

The transparency log. Append-only and tamper-evident: every signature is recorded, so an unexpected signing event is publicly detectable.

Keyless: no long-lived key to steal, and every signature is publicly auditable


Build job
Untrusted runner
Build output — data only
Trusted control plane
Outside the runner blast radius
Provenance generation
Digest, source ref, SBOM, pipeline metadata
Signing infrastructure
Out-of-band — build cannot reach it
Storage + verification
in-toto attestations alongside OCI artifacts
Verify before deploy
Deploy-time verification
CI gates · CLI · deploy controls
Untrusted runner
Trusted control plane

How GitLab does SLSA L3: gate, generate, sign, store

1

Gate

Rails checks eligibility: flag on, public project, build stage, attest vars.

2

Generate

Rails builds the provenance after the job — source, git commit, runner details.

3

Sign

Platform OIDC token; cosign attest-blob → Fulcio cert → Rekor log.

4

Store

Saved to slsa_attestations; signed bundle in object storage (2-year expiry).

5

Verify

Consumer runs glab attestation verify; cosign checks cert, Rekor, identity.

The Rails backend — not the CI job — is the only thing that can sign: a compromised build never sees the OIDC token. That's what makes provenance unforgeable.

Level 3 succeeds or fails on multi-tenant isolation

What must be true

Ephemeral runners that tear down to a trustworthy baseline.

Runner identity attested to the control plane.

Hermetic or tightly-controlled dependency resolution.

Content-addressed caching with integrity guarantees.

Sigstore stack

Cosign for signing and verification, Fulcio for cert issuance, Rekor for transparency — with room for managed or customer-controlled trust.

What auditors care about

Can a malicious tenant sign?

Can a reused runner leak state?

Can a verifier reject suspect infrastructure?

The cost is real, but manageable when you tune the trust path

Pipeline overhead

Naive: 30–90s of added pipeline time.

Tuned: ~15s.

Levers: content-addressed caching, parallel attestation, async signing.

Developer friction

New failure modes appear immediately: missing attestations, signature verification errors, and hermetic-build violations.

Rollout pattern

Feature-flag the capability, onboard by team, and embed a security partner for the first wave. Container pipelines tolerate the overhead; tight monorepo inner loops are where every second becomes political.

SLSA is strong against build-path tampering, not every risk

Catches well

Forged or unsigned artifacts.

Unexpected build environment identity.

Attestation mismatch between source, artifact, and pipeline record.

Policy violations at deploy time when verification is enforced.

Helps investigate

Suspicious dependency resolution paths.

Unexpected rebuilds or replay attempts.

Incident triage when provenance evidence exists.

Audit and procurement evidence requests.

Does not solve alone

Malicious but properly-built source code.

Business-logic flaws and runtime exploits.

Weak secrets hygiene outside the build path.

Third-party risk if verification is never enforced downstream.

Customer reality is what forces the architecture to mature

The technical implementation is only part of the program; onboarding, audit evidence, and escalation playbooks consume the rest.

RFP pattern Customers often ask for Level 3 in the RFP, accept Level 2 initially, then reopen the gap at renewal.

Regulated buyers Government and regulated buyers are far less flexible when deploy gates depend on verifiable provenance.

Real differentiation Most platforms can generate attestations. What matters is where trust is rooted and how verification is enforced as one path.

The fastest path to Level 3 is sequencing the hard problems

0

Inventory

Map artifact types, runners, registries, and current trust roots.

1

Generate provenance

Start with broad coverage and portable attestation formats.

2

Move signing out-of-band

Do this before hermetic-build work, or you harden the wrong boundary.

3

Attest runner identity

Prove where the build ran, not just what it emitted.

4

Enforce verification

CI gates, CLI, admission control, and deploy policy make provenance useful.

Prefer boring standards: Sigstore, in-toto, OCI artifacts, and verifiers that don't lock you to one cloud or one proprietary stack.

Gitlab Incremental Approach to SLSA L3

Three bottom lines and one ask

Take-home: if a fully compromised build job can still sign a trusted attestation, you are not at Level 3.

Architecture

L3 moves the trust boundary: sign out-of-band, isolate runners, enforce verification. Not “more YAML.”

Sequencing

Inventory, then provenance, then out-of-band signing, then runner identity, then enforcement. Order beats effort.

Buyer pull

Provenance now shows up in RFPs and at renewal. Ship it as one platform path, not compliance theater.

OWASP GLOBAL AppSec

VIENNA'26 JUN 25-26

25 years of open source security

THANK YOU!