THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
NORTH AMERICA

Embedded Linux Conference

Verification Toward Applying SLSA in Automotive IVI Software Development

Yuta Kiyoumi & Takashi Ninjouji
Honda R&D

Who Are We?

- Yuta Kiyoumi
  - Assistant Chief Engineer / Security Architect, IVI Software & OSPO @ Honda

- Takashi Ninjouji
  - Chief Engineer & OSPO @ Honda

Agenda

1. Honda Motivation: Why SLSA
2. Applying SLSA to AAOS-based IVI
3. Lessons learned

Honda Motivation : Why SLSA?

Automotive IVI: A Multi Layer Software Supply Chain

The OEM integrates source, binary and build outputs from multiple organizations.

Visibility gaps that matter to risk management

- Binary only deliveries
- Different build systems
- Hidden dependency paths
- Frequent updates

Regulation, Standards, Evidences

Regulations

UN R155
Cybersecurity Management System (CSMS) for type approval.

EU CRA
Reg. (EU) 2024/2847.
Secure product lifecycle.

Process standards

ISO/SAE 21434
Lifecycle cybersecurity engineering for vehicle E/E systems.

OpenChain: ISO/IEC 5230, 18974
5230: Open source licence compliance.
18974: Open source security assurance.

NIST SSDF (Sp 800-218)
Secure software development framework.

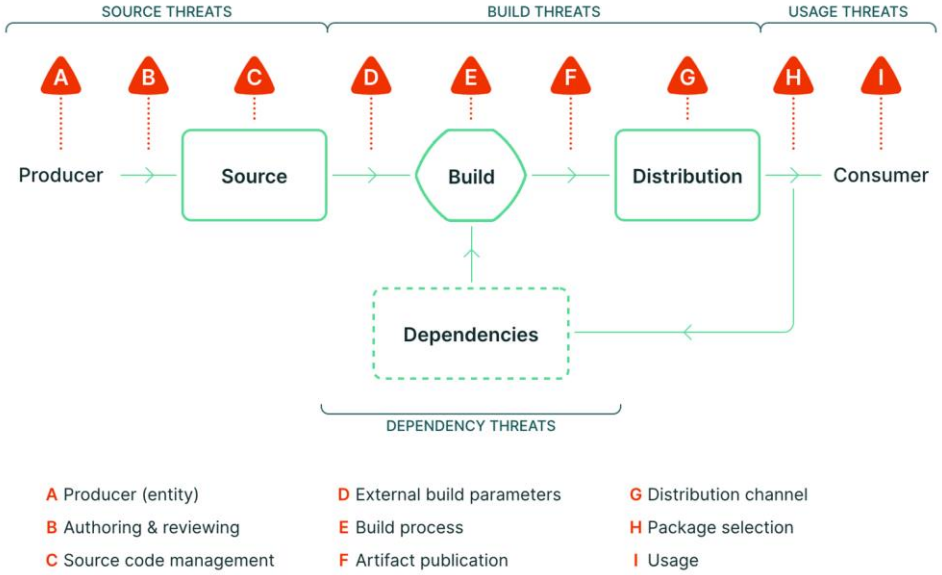
Artifact Evidence

SBOM
SPDX (ISO/IEC 5962), CycloneDX (ECMA-424).
The component inventory.

SLSA 1.2
Build + source tracks.
Tamper-evident, machine-verifiable claims about each artifacts.

SLSA: Supply-chain Levels for Software Artifacts

A security framework, a checklist of standards and controls to prevent tampering, improve integrity, and secure packages and infrastructure.



Build & Source Track

A verifiable evidence, cryptographic, layer for source and build integrity

Build Track

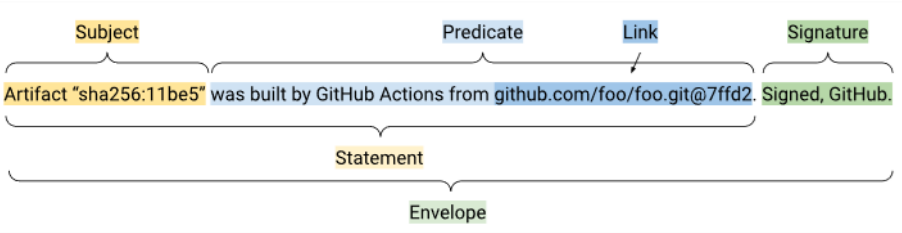
L1	Provenance showing how the package was built
L2	Signed provenance, generated by a hosted build platform
L3	Hardened build platform

Source Track

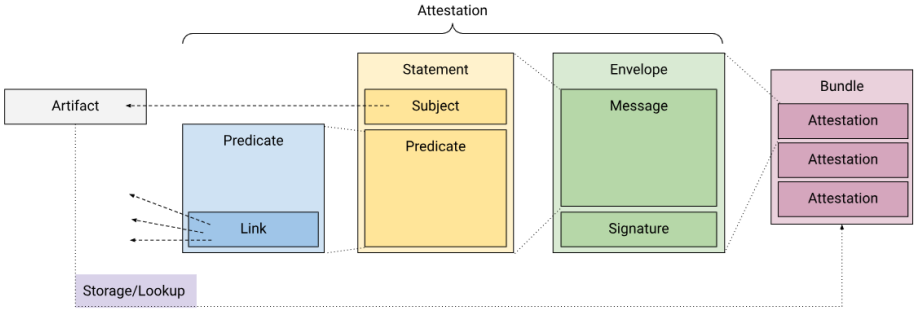
L1	Use a version control system
L2	Preserve Change History and generate Source Provenance
L3	Enforce organizational technical controls
L4	Require code review

Attestation

A software attestation is an authenticated statement (metadata) about a software artifact or collection of software artifacts. The primary intended use case is to feed into automated policy engines, such as in-toto and Binary Authorization.



Component	Recommendation
Envelope	**DSSE** (ECDSA over NIST P-256 (or stronger) and SHA-256.)
Statement	**in-toto attestations**
Predicate	Choose as appropriate, i.e.; **Provenance**, **SPDX**, **other predicates defined by third-parties**.
Component	Recommendation
Envelope	**DSSE** (ECDSA over NIST P-256 (or stronger) and SHA-256.)



SBOM & SLSA

Enabling High Confidence Risk Analysis

SBOM

Inventory view
(composition transparency)

Which components are included?
Which versions and licenses?
Which CVEs or EOL components?
Which supplier should be contacted?

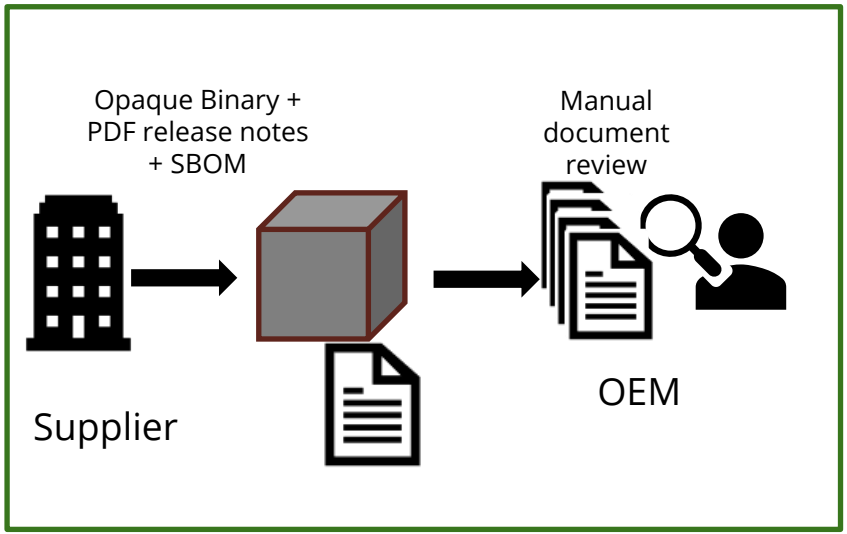
SLSA

Provenance view
(build integrity evidence)

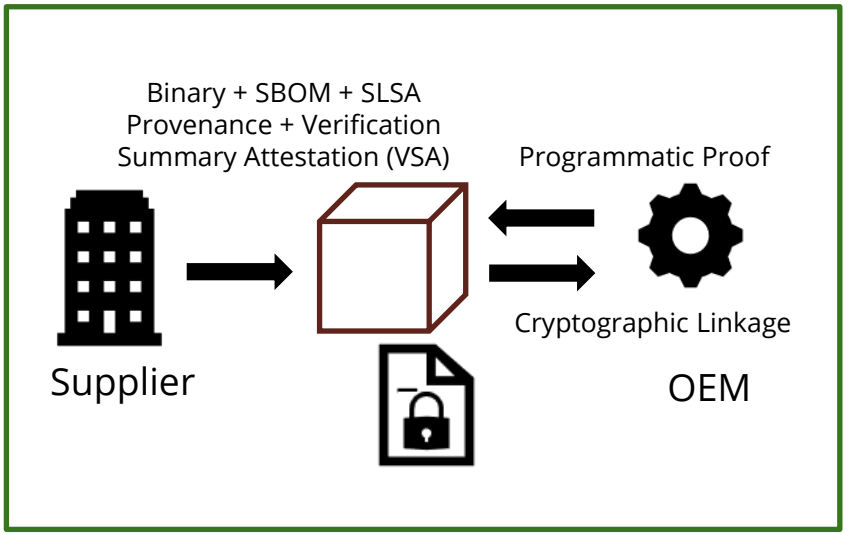
Which source revisions were used?
Which build workflow ran
Which builder identity produced it?
Can the claim be verified automatically?

From Document To Machine Verifiable Evidence

Before (Trust-based)



After (Evidence-based)



Can SLSA work for AOSP-scale IVI?

Applying SLSA to AAOS-based IVI

THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
NORTH AMERICA
Embedded Linux Conference

How to apply SLSA

- There are many types of automotive software and different ways to develop them (like build flows and source code management).
- We first performed an initial validation and performance evaluation in AAOS.
- To prepare for the future, we chose methods that can work in any environment.

Why important for IVI Software Development

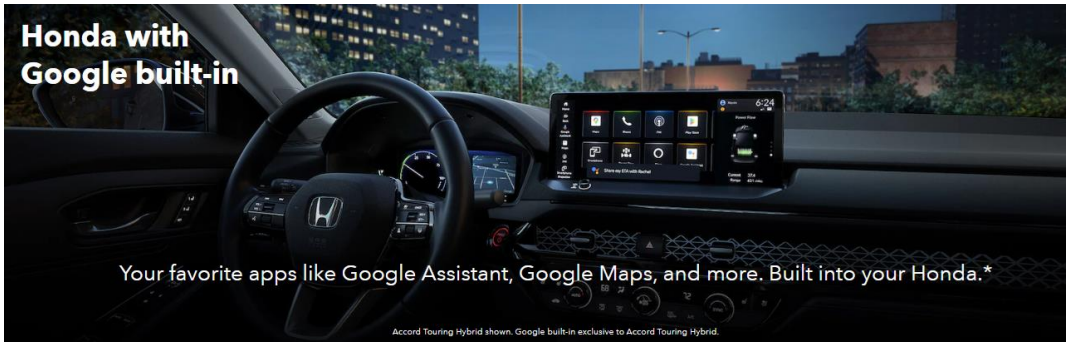
- IVI software is large-scale and relies on a complex supply chain. It continuously evolves through OTA updates.

- On the other hand, risks like bad code or system tampering are increasing. Also, we must follow more laws and standards than before.

- Therefore, we evaluate SLSA as a framework. This helps us assess whether SLSA can support evidence-based supply chain assurance.

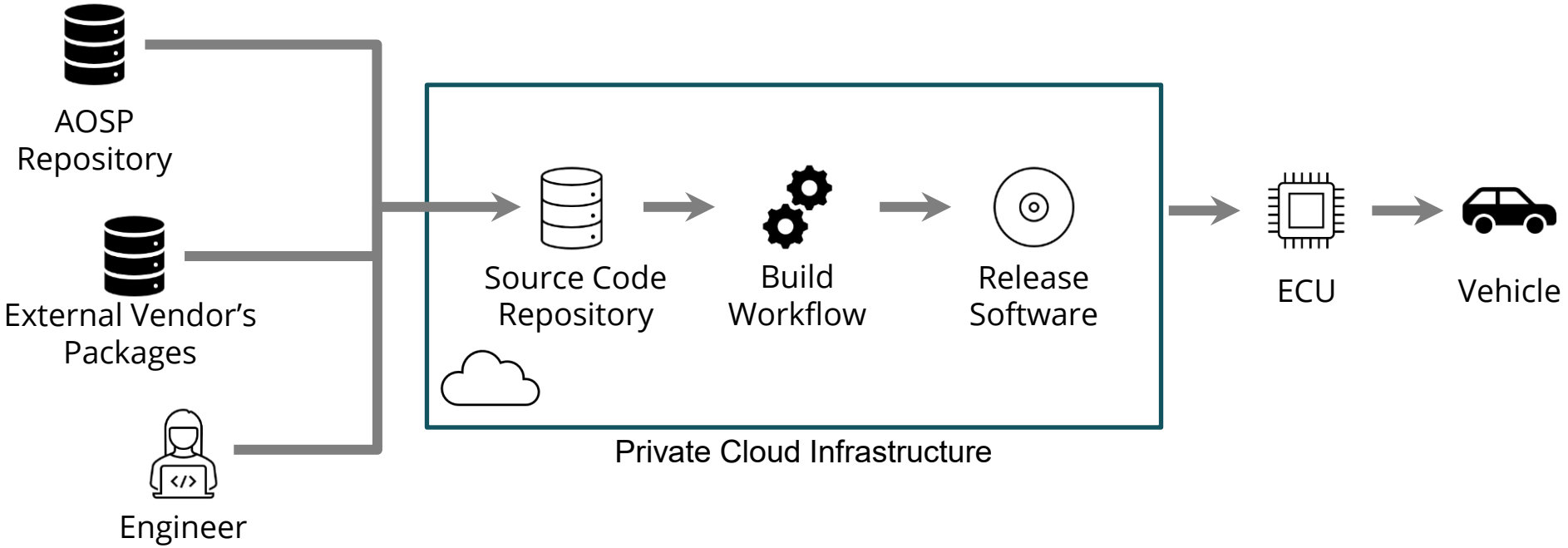
About AAOS

- Android Automotive OS
- The First-Ever Honda with Google built-in.
  - The 2023 Accord Touring Hybrid with Google built-in also offers an electrifying hybrid powertrain, 12-speaker Bose premium sound system, Head-Up Display, and more.



2023 Honda Accord Google Services - YouTube

Software Production Flow



How to proceed with SLSA evaluation

- Build Track
- Source Track
  -> Generate and verify the Attestation

Attestation framework

- Dependency Track
- Build Environment Track
  -> Not included in the current verification as the specifications were still draft.

Build Flow of AAOS

```
(aosp_sync.sh):
repo init --partial-clone -b main -u https://android.googlesource.com/platform/manifest

repo sync -c -j8
```

```
(aosp_build.sh):
source build/envsetup.sh

lunch aosp_cf_x86_64_auto-trunk_staging-userdebug

m
```

Note: This command represents the AAOS build flow, the actual builds process varies by chip vendor.

https://source.android.com/docs/setup/start

Attempt to generate Build Provenance

- The following command was executed:

```
> in-toto-run ¥
  --step-name “aosp-build“ ¥
  --signing-key key.pem ¥
  --materials "${ANDROID_DIR}" ¥
  --products "${ANDROID_DIR}/out" ¥
  --record-streams ¥
  --run-timeout 18000 ¥
  -- aosp_build.sh
```

=> But, the process remained pending.

Why is pending?

- Specifying directories with --materials and --products causes all files to be recursively scanned within those directories and perform hash calculations on all of them.

- In large-scale software development, specifying the entire source code or deliverables can overwhelm the hash value calculation process required for Provenance generation.

```
ubuntu@server:~/aosp$ find /home/ubuntu/aosp/ -type f | wc -l
1641570
ubuntu@server:~/aosp$ find /home/ubuntu/aosp/  | wc -l
1963847
```

```
if isdir(path):
    for base, dirs, names in os.walk(
        path, followlinks=self._follow_symlink_dirs
    ):
        # Filter directories to avoid unnecessary recursion below
        # NOTE: Normalize to filter on directory paths without
        # their dot-slash prefix, if path was dot.
        dirs[:] = [
            dirname
            for dirname in dirs
            if not self._exclude(normpath(join(base, dirname)))
        ]

        for filename in names:
            # NOTE: Normalize to filter on and return file paths
            # without their dot-slash prefix, if path was dot.
            filepath = normpath(join(base, filename))

            if self._exclude(filepath):
                continue

            if not isfile(filepath):
                logger.info(
                    "File '%s' appears to be a broken symlink. "
                    "Skipping...",
                    filepath,
                )
                continue

            name = self._mangle(filepath, hashes, prefix)
            hashes[name] = self._hash(filepath)
```

https://github.com/in-toto/in-toto/blob/develop/in_toto/resolver/_resolver.py

Approach to resolve

- Instead of including all source code and deliverables, select only the necessary files.

Target	File
Materials	Specify revision-locked manifest It is possible to identify which source code was included in the build.
Products	Specify only the final product, excluding intermediate products.

What is Manifest file:
The file that defines the source code to be built.
(Stored repository, revision, etc.)
AOSP retrieves and builds source code based on the Manifest file.

```
repo manifest -r -o revision-locked-manifest.xml
```

```
<project name="platform/packages/apps/Car/CalendarPrebuilt"
path="packages/apps/Car/CalendarPrebuilt"
revision="2a8fc1d4e6ef3c0f1130148133bf6192b207541e"
upstream="refs/heads/aosp.lnx.15.3.r1-rel"
groups="aosp,emulator" />
```

Example manifest

Regenerate the Attestation

```
STARTED_ON=$(date -u +"%Y-%m-%dT%H:%M:%SZ")

in-toto-run ¥
  --step-name “aosp-build“ ¥
  --signing-key key.pem ¥
  --materials /work/materials-manifest/ ¥
  --products ¥
    ${ANDROID_DIR}/out/system.img ¥
  --metadata-directory /work ¥
  --record-streams ¥
  --run-timeout 18000 ¥
  -- aosp-build.sh

FINISHED_ON=$(date -u +"%Y-%m-%dT%H:%M:%SZ")
```

#1: generating link metadata

```
$ python3 generate_slsa_predicate.py ¥
  --linkfile aosp-build.*.link ¥
  --output aosp-build-predicate.json ¥
  --started-on "$STARTED_ON" ¥
  --finished-on "$FINISHED_ON"
```

#2: convert to predicate from link metadata

```
$ cosign attest-blob system.img ¥
  --key cosign.key ¥
  --type https://slsa.dev/provenance/v1 ¥
  --predicate aosp-build-predicate.json ¥
  --bundle aosp-build-attestation.bundle.json
```

#3: Attest the provenance

Provenance (Attestation - Statement)


```
{
  "_type": "https://in-toto.io/Statement/v1",
  "subject": [
    {
      "name": "system.img",
      "digest": { "sha256": "95eee203994b10 .. omitted ..  }
    }
  ],
  "predicateType": "https://slsa.dev/provenance/v1",
  "predicate": {
    "buildDefinition": {
      "buildType": "https://hondabuild.com/ivi-build/v1",
      "externalParameters": {
        "buildCommand": [ "aosp-build.sh" ],
        "stepName": "aosp-build"
      },
      "internalParameters": {
        "environment": {}
      },
      "resolvedDependencies": [
        {
          "uri": "file:///work/materials-manifest/system-manifest.xml",
          "digest": { "sha256": "23eb4f1f8d865 .. omitted .. " },
          "name": "system-manifest.xml"
        }
      ]
    },
```

```
    "runDetails": {
      "builder": {
        "id": "https://hondabuild/ivi-builder/v1"
      },
      "metadata": {
        "invocationId": "aosp-build- 2026-04-23T08:14:04.545345+00:00",
        "startedOn" : "2026-04-23T08:14:04.545345+00:00",
        "finishedOn": "2026-04-23T09:00:40.125811+00:00"
      },
      "byproducts": [
        {
          "uri": "data:application/json; base64,buildMetadata",
          "digest": { "sha256": "2d7473f7e8a7869097a87865 .. omitted .. " },
          "name": "buildMetadata",
          "mediaType": "application/json",
          "content": "eyJyZXR1cm4tdmFsdWUiOiAwLCAic3R .. omitted .. "
        }
      ]
    }
  },
  "_signatures": [
    {
      "keyid": "1e61a9a6fa9c9a86257a9904512d4ac465 .. omitted .. ",
      "sig": ... omitted ...
    }
  ]
}
```

Source Track

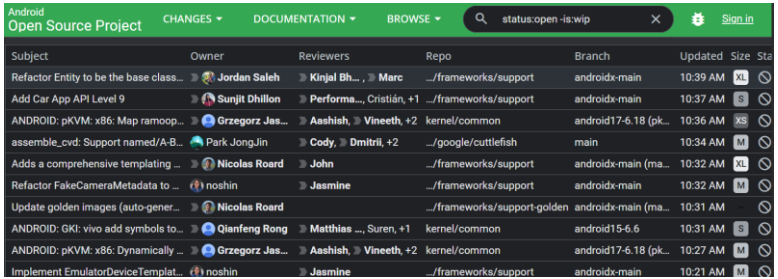
How to generate Source Track Attestation for AAOS

- Attempted verification based on Gerrit, the SCS of AAOS.

Gerrit

Gerrit is a web-based code review system for projects that use Git. Gerrit encourages a more centralized use of Git by allowing all authorized users to submit changes, which are automatically merged if they pass code review. In addition, Gerrit simplifies reviewing, displaying changes side by side in the browser and enabling inline comments.

https://source.android.com/docs/setup/download/source-control-tools



https://android-review.googlesource.com/q/status:open+-is:wip

Source Track

How to generate Source Track Attestation for AAOS

- What should be the Subject?
  - AAOS has a large number of repositories and a vast amount of merge history.
  - This verification focused only on latest changes merged in the software releases of each repository.

Source Track

Generate Source Provenance

1. Retrieve software release revision information from the manifest file.

1. Retrieve merge information from Gerrit based on the revision information.
   - revision ID / parent revision / branch / contributor / review / submit

```
curl -s -u "$USER:$PASS" -L
"https://$GERRIT_HOST/a/changes/?q=project:$ENCODED_PROJECT+commit:$COMMIT_HA
SH&o=DETAILED_LABELS&o=MESSAGES&o=DETAILED_ACCOUNTS&o=CURRENT_REVISION&o=CURR
ENT_COMMIT" ¥
  | sed '1d' ¥
  | jq --arg project "$PROJECT" --arg host "$GERRIT_HOST" '.[0] as $base |
$base.current_revision as $rev | {
  "_type": "https://in-toto.io/Statement/v1",
  "subject": [
    {
      "name": ($project + "@" + $base.branch),
      "digest": {
        "sha1": $rev
      }
    }
  ],
  "predicateType": "https://slsa.dev/vcs-provenance/v0.1",
  "predicate": {
    "repository": {
      "uri": ("https://" + $host + "/" + $base.project),
      "branch": $base.branch,
      "type": "gerrit"
    },

... omitted ...
```

Source Track

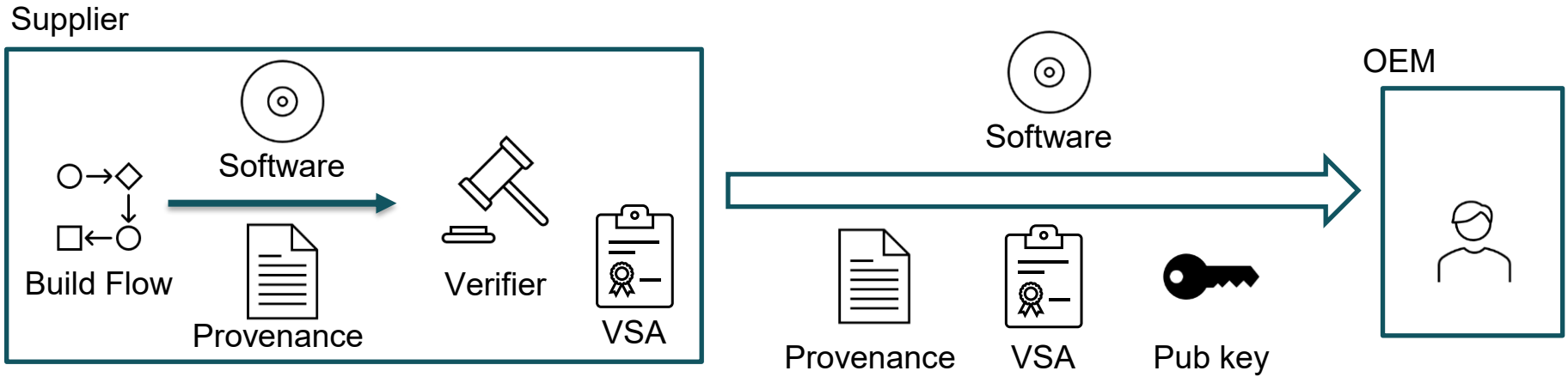
Source Provenance


```
{
  "_type": "https://in-toto.io/Statement/v1",
  "subject": [
    {
      "name": "platform/packages/services/Car@release-for-ABC",
      "digest": {  "sha1": "7c1afe84a2d7c6761d6a712119c55ad2795d6907" }
    }
  ],
  "predicateType": "https://slsa.dev/vcs-provenance/v0.1",
  "predicate": {
    "repository": {
      "uri": "https://scs-honda.com/platform/packages/services/Car",
      "branch": "release-for-ABC",
      "type": "gerrit"
    },
    "revision": {
      "author": {
        "name": "Taro Honda",
        "email": "taro.honda@honda",
        "timestamp": "2026-04-29T18:30:21Z"
      },
      "committer": {
        "name": "Alex Rivera",
        "email": "a.rivera@honda",
        "timestamp": "2026-04-29T18:35:10Z"
      },
      "message": "[PRJ-123] bugs fix¥n",
      "parents": [ "9196b3c36d38914a6adab89b94b030d6cb373fb4" ]
    },
```

```
"change": {
      "id": "platform/packages/services/Car~157306",
      "changeId": "I9a4c17dcc7b6f094a935f63440e8e362ad6c8721",
      "number": 157306,
      "subject": "ex - update function",
      "status": "MERGED",
      "createdOn": "2026-04-15T05:44:45Z",
      "submittedOn": "2026-04-15T08:24:58Z",
      "uri": "https://scs-honda.com/c/platform/packages/services/Car/+/157306",
      "review": {
        "approvals": [
          {
            "label": "Code-Review",
            "value": 0,
            "approver": {
              "name": "Hanako Honda",
              "email": "hanako.honda@honda",
            }
          },
          {
            "label": "Verified",
            "value": 0,
            "approver": {
              "name": "Jiro Honda",
              "email": "j.honda@honda",
            }
          }
        ]
```

Verify Attestation

- Evaluation in a scenario expected for an automotive OEM:
  - The OEM verifies the product built by the supplier.



How to Verify using Attestation

Step 1: Check SLSA Build level

First, check the SLSA Build level by comparing the artifact to its provenance and the provenance to a preconfigured root of trust. The goal is to ensure that the provenance actually applies to the artifact in question and to assess the trustworthiness of the provenance. This mitigates some or all of **threats** "E", "F", "G", and "H", depending on SLSA Build level and where verification happens.

Given an artifact and its provenance:

1. **Verify** the envelope's signature using the roots of trust, resulting in a list of recognized public keys (or equivalent).
2. **Verify** that statement's `subject` matches the digest of the artifact in question.
3. Verify that the `predicateType` is `https://slsa.dev/provenance/v1`.
4. Look up the SLSA Build Level in the roots of trust, using the recognized public keys and the `builder.id`, defaulting to SLSA Build L1.

https://slsa.dev/spec/v1.2/verifying-artifacts

How to Verify using Attestation

```
(Attestation integrity (signature verification) )
cosign verify-blob-attestation ¥
  --key cosign.pub ¥
  --bundle aosp-build-attestation.bundle.json ¥
  system.img
```

```
(Check predicateType)
PAYLOAD=$(jq -r '.dsseEnvelope.payload' "$BUNDLE_FILE" | base64 -d)
if [ "$(echo "$PAYLOAD" | jq -r '.predicateType')" = "https://slsa.dev/provenance/v1" ];
then
    // OK
else
    // NG
fi
```

What is the Next Step?

- Detailed Verification of Provenance
  - In cases of Black-box development, alignment between the OEM and the supplier is necessary.

Step 2: Check expectations

Next, check that the package's provenance meets your expectations for that package in order to mitigate threat "D".

In our t… registry… access… this ad…

What	Why
Builder identity from Step 1	To prevent an adversary from building the correct code on an unintended platform
Canonical source repository	To prevent an adversary from building from an unofficial fork (or other disallowed source)
`buildType`	To ensure that `externalParameters` are interpreted as intended
`externalParameters`	To prevent an adversary from injecting unofficial behavior

Step 3: (Optional) Check dependencies recursively

Finally, recursively check the `resolvedDependencies` as available and to the extent desired. Note that SLSA v1.0 does not have any requirements on the completeness or verification of `resolvedDependencies`. However, one might wish to verify dependencies in order to mitigate dependency threats and protect against threats further up the supply chain. If `resolvedDependencies` is incomplete, these checks can be done on a best-effort basis.

https://slsa.dev/spec/v1.2/verifying-artifacts

Source Track

How to Verify using Attestation

- The Build Track verifies Provenance.
- The Source Track verifies VSA.

Step 1: Check the SCS

First, check the SLSA Source level by comparing the artifact to its VSA and the VSA to a preconfigured root of trust. The goal is to ensure that the VSA actually applies to the artifact in question and to assess the trustworthiness of the VSA. This mitigates threats within “B” and “C”, depending on SLSA Source level.

https://slsa.dev/spec/v1.2/verifying-source

What is VSA

- Verification Summary Attestation (VSA) summarizes the results of verifying an artifact against SLSA requirements and organizational policies.

- This verification assumes that the VSA was created as a result of verifying Provenance within the supplier.


```
{
  "_type": "https://in-toto.io/Statement/v1",
  "subject": [
    {
      "name": "git+https://scs-honda.com/platform/packages/services/Car@release-for-
ABC",
      "digest": { "sha1": "7c1afe84a2d7c6761d6a712119c55ad2795d6907" },
      "annotations": {
        "sourceRefs": ["refs/heads/release-for-ABC"]
      }
    }
  ],
  "predicateType": "https://slsa.dev/verification_summary/v1",
  "predicate": {
    "verifier": {
      "id": "https://scs-honda.com/source-verifier/v1",
      "version": { "source-vsa-verifier": "v0.1.0" }
    },
    "timeVerified": "2026-04-27T00:00:00Z",
    "resourceUri": "git+https://scs-honda.com/platform/packages/services/Car",
    "policy": {
      "uri": "https://scs-honda.com/policies/source-release-policy/v1",
      "digest": { "sha256": "1234...REPLACE..." }
    },
    "inputAttestations": [
      {
        "uri": "https://scs-honda.com/attestations/sourceprovenance_platform-
car_123456.intoto.jsonl",
        "digest": { "sha256": "abcd...REPLACE..." }
      }
    ],
    "verificationResult": "PASSED",
    "verifiedLevels": ["SLSA_SOURCE_LEVEL_2"],
    "slsaVersion": "1.2"
  }
}
```


Example VSA

Source Track

How to Verify using Attestation

- VSA verification first checks the basic integrity of the attestation:
  - Verify the signature of the VSA envelope
  - Confirm that the subject identifies the expected source revision or source artifact
  - ...
- The Source Track needs to define roots of trust in advance, including:
  - Which SCS identities are trusted
  - Which VSA verifier identities are recognized
  - Up to which SLSA Source Level each SCS/verifier combination is trusted
  - ...

=> Future work: Source Track trust and policy validation.

Verification MUST include the following steps:

1. Verify the signature on the VSA envelope using the preconfigured roots of trust. This step ensures that the VSA was produced by a trusted producer and that it hasn't been tampered with.
2. Verify the statement's `subject` matches the digest of the artifact in question. This step ensures that the VSA pertains to the intended artifact.
3. Verify that the `predicateType` is `https://slsa.dev/verification_summary/v1`. This step ensures that the in-toto predicate is using this version of the VSA format.
4. Verify that the `verifier` matches the public key (or equivalent) used to verify the signature in step 1. This step identifies the VSA producer in cases where their identity is not implicitly revealed in step 1.
5. Verify that the value for `resourceUri` in the VSA matches the expected value. This step ensures that the consumer is using the VSA for the producer's intended purpose.
6. Verify that the value for `verificationResult` is `PASSED`. This step ensures the artifact is suitable for the consumer's purposes.
7. Verify that `verifiedLevels` contains the expected value. This step ensures that the artifact is suitable for the consumer's purposes.

https://slsa.dev/spec/v1.2/verification_summary

Result to verify

- We were able to (to some extent) mechanically verify that the software released by the Tier-1 supplier was through the proper procedures.
  - Have the deliverables provided by the supplier been tampered with?
  - Does the build flow meet certain standards?
  - Is the software from approved source code?

- The SLSA-defined levels now allow for a unified standard to evaluate the level of security strength.

Lessons Learned

Key Insights

- **Technical Challenges:**
  - Large-scale builds require scoped provenance (e.g., manifest-based filtering in AAOS)
  - Hash calculation overhead must be managed

- **Process Improvements:**
  - Pre-alignment with suppliers on SLSA requirements is critical
  - Automated attestation generation standardizes the process, improves reproducibility

- **Regulatory Feasibility:**
  - SLSA provides artifact-level evidence complementary to Regulatory
  - Enables machine-verifiable supply chain transparency

Future works

- Clarification of OEM acceptance conditions
- Developing SLSA application model for multi-layer software supply chains
- Advancing automation through toolchains
- Evaluating the Synergy between SLSA and SBOM

Q&A

Thanks!