KubeCon | CloudNativeCon

North America 2024

KubeCon | CloudNativeCon
North America 2024

Practical Supply Chain Security: Implementing SLSA Compliance from Build to Runtime

Enguerrand Allamel - Ledger

WHOAMI

Enguerrand Allamel

Academic Experience: Tsinghua University (Beijing) and Epitech (Paris)

Current Role: Staff Cloud Security Engineer at Ledger with a focus on Supply Chain Security topics and Cloud Security

Company Overview: Ledger specializes in secure hardware wallets and cutting-edge security products

LEDGER

Agenda

1. Why is **Supply Chain Security Important**?
2. What is **SLSA** (**S**upply-chain **L**evels for **S**oftware **A**rtifacts)?
3. **Example Implementation**
    a. On the **Build** Side
    b. On the **Runtime** Side
4. Going Further with **HSM** (**H**ardware **S**ecurity **M**odule)

Why is Supply Chain Security Important ?

61% of U.S. businesses were directly impacted by software supply chain attacks between April 2022 and April 2023*

Types of attack	Know examples
Artifact Repository Compromis	**lottie-player:** Malicious code was injected into a popular JavaScript library. This compromised code introduced unauthorized Web3 wallet connection prompts on websites using the library. - **Financial Loss**: At least one user reportedly lost 10 Bitcoin ($723,436). - **Wide Reach:** Over 4 million lifetime uses and 94,000 weekly downloads
Compromise build process	**SolarWinds:** Attackers infiltrated the build platform and deployed a malicious implant that injected unauthorized behavior into each build. - **Massive Data Breach:** Approximately 18,000 organizations were affected. - **Financial Loss:** SolarWinds experienced a 40% decline in stock price.

**Source: Leader's Guide to Software Supply Chain Security, 2024*

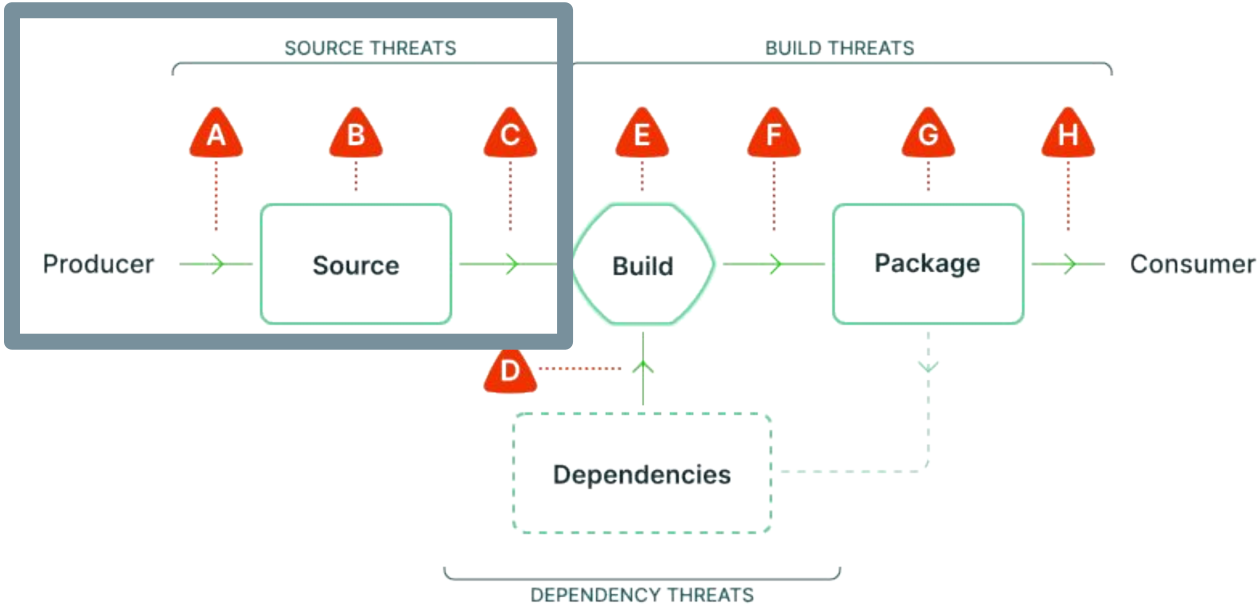
What is SLSA?

- **SLSA: S**ecurity **L**evels for **S**oftware **A**rtifacts
- **Backing:** Sponsored by the OpenSSF (Open Source Security Foundation), associated with the Linux Foundation
- **Collaborative Framework:** Developed through cross-industry collaboration
- **Purpose:** Establishes standards and guidelines for securing software supply chains
- **Core Components:**
  - SLSA Requirements
  - SLSA Provenance (similar to attestation)
- **Audience:** Tailored for software producers, consumers, and infrastructure providers

Website:
https://slsa.dev/

Github Repository:
https://github.com/slsa-framework/slsa

Scope of Threats and Attack in SLSA: Source



Example:

- Code modification within a Git repository
- Permission bypass on a Git repository hosting platform (e.g., GitLab, GitHub, Gitea)

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

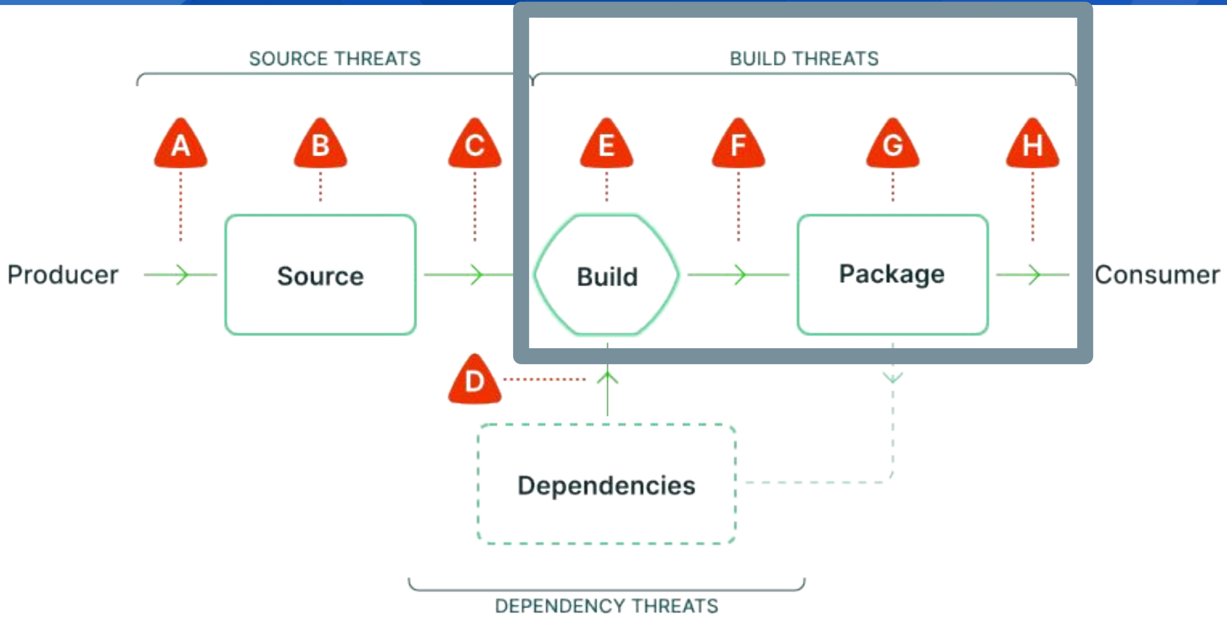
F Upload modified package

G Compromise package registry

H Use compromised package

Diagram based on SLSA 1.0 documentation

Scope of Threats and Attack in SLSA: Build



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

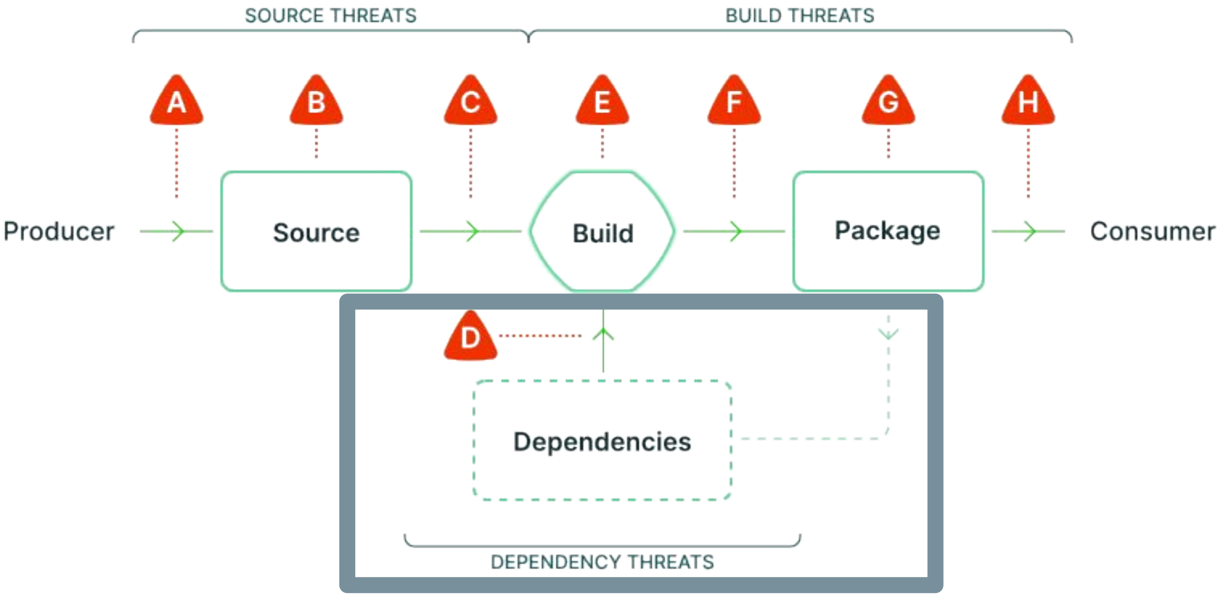
H Use compromised package

Example:

- CI/CD or build platform compromised
- Package registry compromised

Diagram based on SLSA 1.0 documentation

Scope of Threats and Attack in SLSA: Dependency



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

Example:

- Typosquatting of a package in dependencies hosted on platforms like PyPI.org, npmjs.com, etc.
- Malicious code embedded within dependencies

Diagram based on SLSA 1.0 documentation

SLSA: Security Levels

Definition of Security Level link to Build Thread of SLSA

Target complexity	Level	Requirements	Focus
By default	**Build L0**	*(none)*	*(n/a)*
Easy	**Build L1**	Provenance showing how the package was built	Mistakes, documentation
Easy-Medium	**Build L2**	Signed provenance, generated by a hosted build platform	Tampering after the build
Hard	**Build L3**	Hardened build platform	Tampering during the build

The SLSA framework in version 1.0 defines levels only for build threats/tracks.

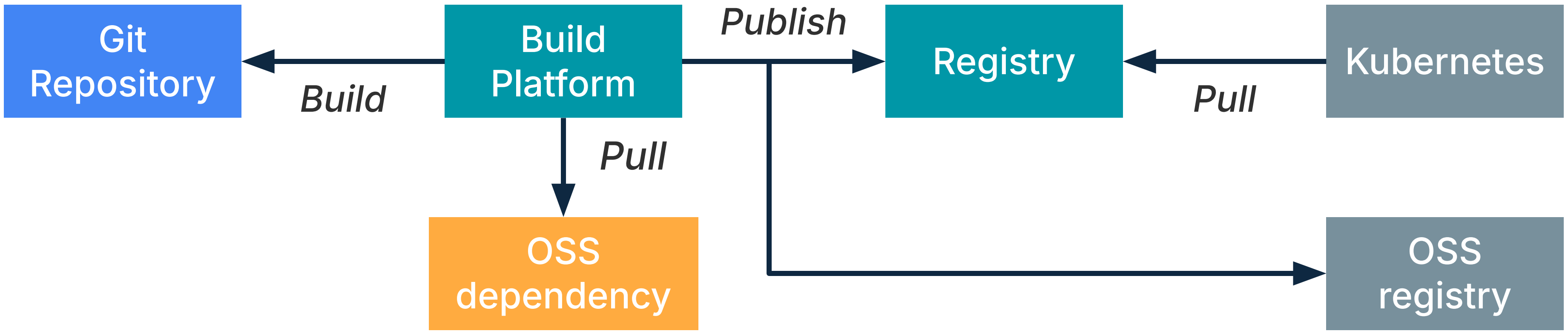
Table based from https://slsa.dev/spec/v1.0/levels

Examples implementation: Context

For this implementation, we assume the following setup:

- The application is in Javascript
- Build platform is Github Action
- Open Source (OSS) npm package from npmjs.com are used
- Registry is Github Package Container (OCI)
- Applications container are running on Kubernetes
- The application is made available open source on npmjs.com

Github Lab

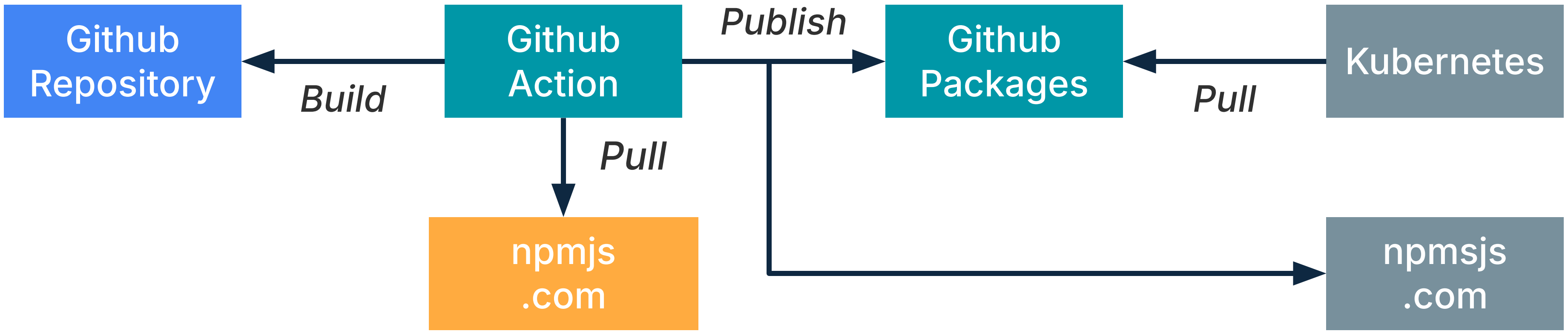


Examples implementation: Context

For this implementation, we assume the following setup:

- The application is in Javascript
- Build platform is Github Action
- Open Source (OSS) npm package from npmjs.com are used
- Registry is Github Package Container (OCI)
- Applications container are running on Kubernetes
- The application is made available open source on npmjs.com

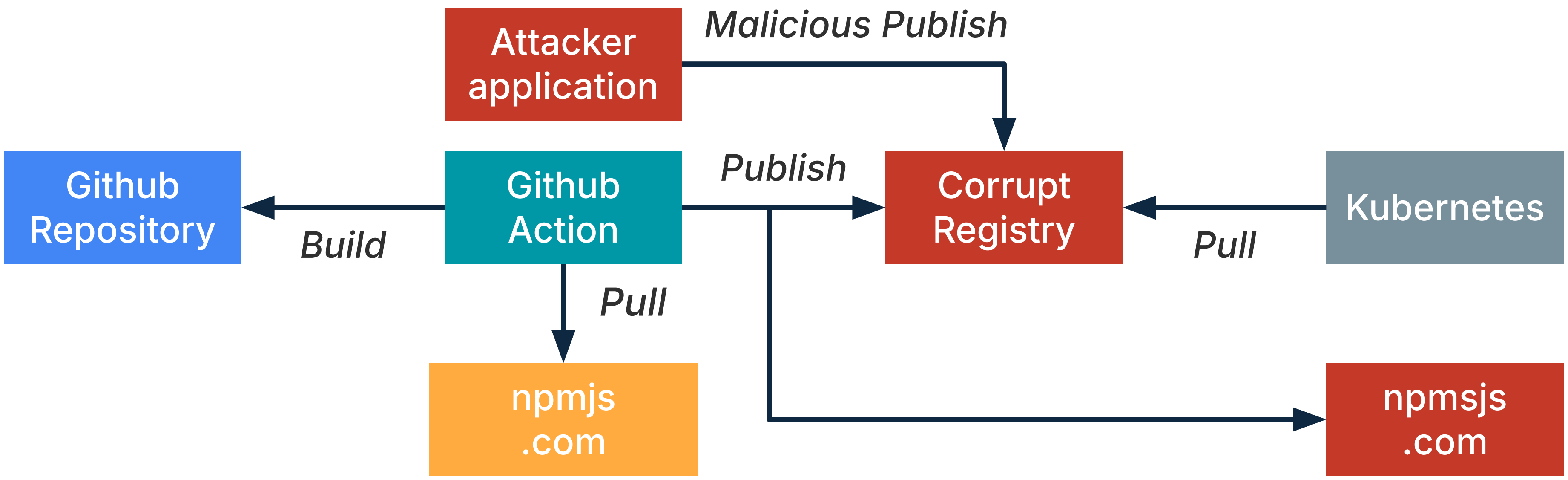
Github Lab



Build Side: Possible defense

How to ensure that the software deployed on Kubernetes was built within GitHub Actions?

- **Signature**: Use tools like Cosign, Notary, etc.
- **Provenance**: Implement SLSA Provenance, In-Toto Attestation, etc.



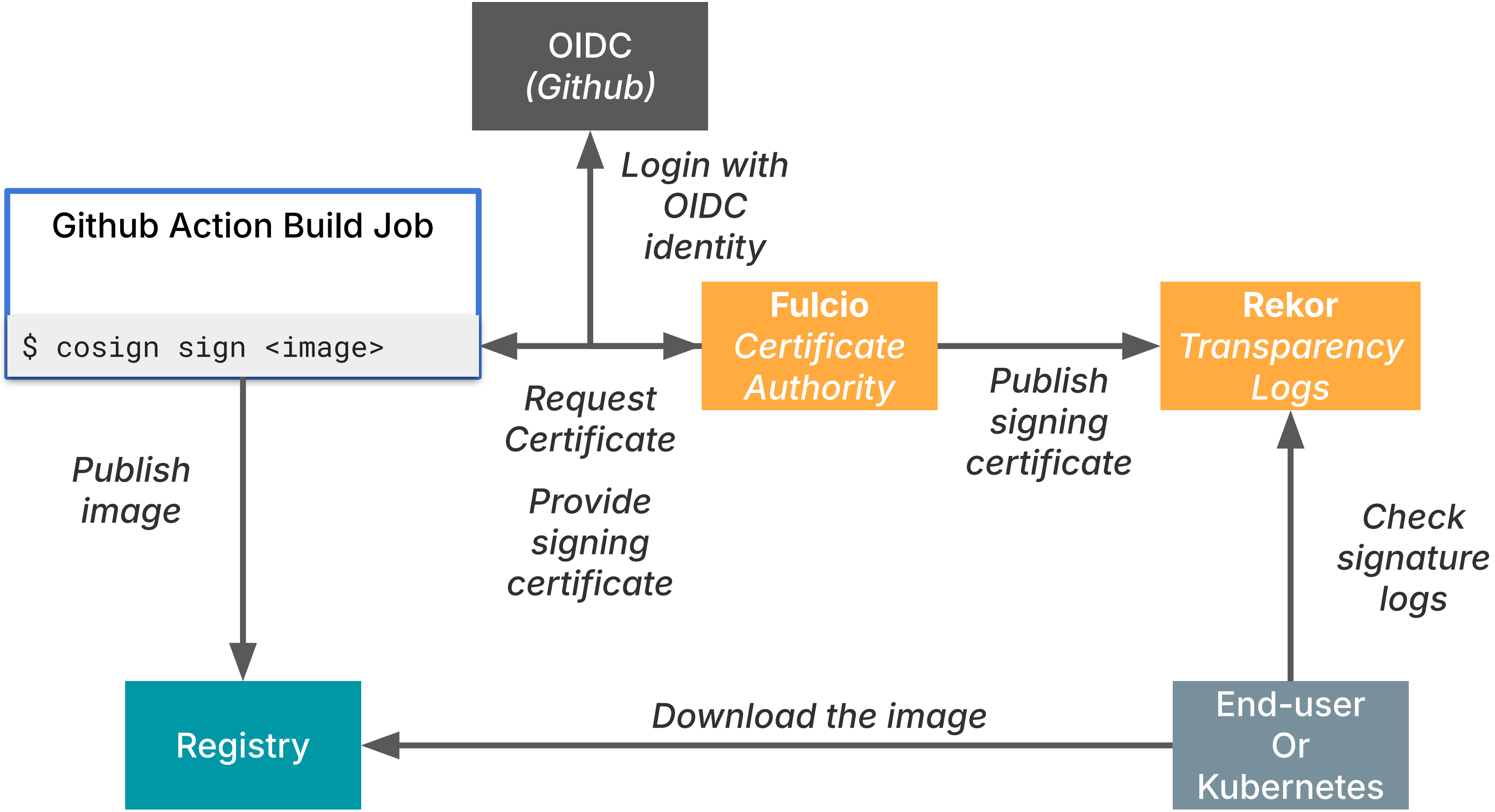
Build Side: SigStore

- **Sigstore:** Open source project for Software Supply Chain Security
- **Backing:** Sponsored by the OpenSSF (Open Source Security Foundation)
- **Purpose:** Provides a simple and secure way to sign software artifacts
- **Motto:** "Sign, Verify, Protect"
- **Core Functions:** Signature of Artifacts, Verification and Monitoring of Signatures
- **Supported Formats:** Works with blobs or container images
- **Tooling Provided:** Cosign: Command-line interface (CLI), Fulcio: Keyless signature authority, Rekor: Transparent metadata logging, etc

Documentation:
https://docs.sigstore.dev/

Github Organisation:
https://github.com/sigstore

SigStore: Keyless signature



SigStore: Rekor log example

sigstore rekor

Rekor Search

Attribute: Log Index

Log Index: 94408140

SEARCH

Showing 1 of 1

Entry UUID: 24296fb24b8ad77afe4b5e11d63661089f0e1166dc8b9cff7154a2084d31ee991d06bbb141e36f6f

TYPE	LOG INDEX	INTEGRATED TIME
intoto	94408140	6 months ago (2024-05-16T19:12:03+02:00)

Hash

sha256:e0a6e04fc087dc693d05f21f50ab6e3c812db2d3f5c3beba5bae5cf85e2897f9

Signature

MEUCIQCuJoaKiLRlYGT+TLbG9KOxzUuU9qy3n6frJXtPa1XrNgIgYAuzrhBKZkDoZSqWl3ewSspsvnIhAn1waJFzbdtCxt4=

Public Key Certificate


```
data:
  Serial Number: '0x301b4d3ebc343aac1c4f5437a4ca1f220520d593'
Signature:
  Issuer: O=sigstore.dev, CN=sigstore-intermediate
  Validity:
    Not Before: 6 months ago (2024-05-16T19:12:02+02:00)
    Not After: 6 months ago (2024-05-16T19:22:02+02:00)
  Algorithm:
    name: ECDSA
    namedCurve: P-256
  Subject:
    extraNames:
      items: {}
    asn: []
```

SigStore: Rekor log example

KubeCon CloudNativeCon North America 2024

```
Subject Alternative Name (critical):
  url:
  - https://github.com/sigstore/sigstore-js/.github/workflows/release.yml@refs/heads/main
OIDC Issuer: https://token.actions.githubusercontent.com
GitHub Workflow Trigger: push
GitHub Workflow SHA: 46e7056ff9912ebfee5298d94024895a9fea76c0
GitHub Workflow Name: Release
GitHub Workflow Repository: sigstore/sigstore-js
GitHub Workflow Ref: refs/heads/main
OIDC Issuer (v2): https://token.actions.githubusercontent.com
Build Signer URI: https://github.com/sigstore/sigstore-js/.github/workflows/release.yml@refs/heads/main
Build Signer Digest: 46e7056ff9912ebfee5298d94024895a9fea76c0
Runner Environment: github-hosted
Source Repository URI: https://github.com/sigstore/sigstore-js
Source Repository Digest: 46e7056ff9912ebfee5298d94024895a9fea76c0
Source Repository Ref: refs/heads/main
Source Repository Identifier: '495574555'
Source Repository Owner URI: https://github.com/sigstore
Source Repository Owner Identifier: '71096353'
Build Config URI: https://github.com/sigstore/sigstore-js/.github/workflows/release.yml@refs/heads/main
Build Config Digest: 46e7056ff9912ebfee5298d94024895a9fea76c0
Build Trigger: push
Run Invocation URI: https://github.com/sigstore/sigstore-js/actions/runs/9116405766/attempts/1
Source Repository Visibility At Signing: public
1.3.6.1.4.1.11129.2.4.2: 04:7a:00:78:00:76:00:dd:3d:30:6a:c6:c7:11:32:63:19:1e:1c:99:67:37:02:a2:4a:5e:b8:de:3c:ad:ff:87:8a:72:80:2f:29:ee:8e:0
```

SigStore: Signature Keyless with Github Action

KubeCon CloudNativeCon North America 2024

- **CI/CD Integration:**
  Within the CI/CD pipeline, specifically during the build job in GitHub Actions, the container image is signed

- **Beyond Signature:**
  A signature alone isn't sufficient, attestation provides additional information to enhance security
  <u>Only prove who signed it</u>

```yaml
...
jobs:
  build-and-push:
    steps:
      - name: Install Cosign
        uses: sigstore/cosign-installer@v3
        ...
      - name: Load Docker metadata
        uses: docker/metadata-action@v5
        ...
      - name: Build and Push container images
        uses: docker/build-push-action@v6
        id: build-and-push
        ...
      - name: Sign the images with GitHub OIDC Token
        env:
          DIGEST: ${{ steps.build-and-push.outputs.digest }}
          TAGS: ${{ steps.docker_metadate.outputs.tags }}
        run: |
          images=""
          for tag in ${TAGS}; do
            images+="${tag}@${DIGEST} "
          done
          cosign sign --yes ${images}
```

Build Side: In-toto (Attestations)

- **In-Toto:** Open source framework for protecting supply chain integrity
- **Backing:** Sponsored by the CNCF
- **Purpose:** Enhances transparency and security in the software supply chain
- **Global Scope:** Focused on supply chain security with integration in multiple languages, primarily Python
- **SLSA Integration:** Can incorporate SLSA Provenance specifications
- **Detailed Attestations:** Provides critical supply chain information, such as code testing results or code review attestations

in-toto

Website:
https://in-toto.io/

Demo (global project):
https://github.com/in-toto/demo

Attestation spec:
https://github.com/in-toto/attestation/tree/v1.0/

In-Toto: Attestation example

- **Predicate File:** Metadata or information embedded in the attestation
- **Example:** Test results, runner details, build environment, etc.
- **SigStore Integration:** Cosign can create and sign predicate files, similar to how it handles containers or blobs
- **Enhanced Security:** Provides trusted information to software consumers
- **Example:** Prove that tests have passed or that code has been reviewed

```json
{
  "_type": "https://in-toto.io/Statement/v0.1",
  "predicateType": "https://cosign.sigstore.dev/attestation/v1",
  "subject": [
    {
      "name": "ghcr.io/ledgerhq/signed-image",
      "digest": {
        "sha256": "<image-sha256>"
      }
    }
  ],
  "predicate": {
    "<my-data>": "<my-value>",
    "Timestamp": "2021-08-11T14:51:09Z"
  }
}
```

```
$ cosign attest --predicate <file> <image>
$ cosign verify-attestation <image>
```

NPMJS.com: Open Source NPM package

KubeCon CloudNativeCon North America 2024

@aenguerrand/kubecon-cloudnativecon-na-2024-supply-chain-security-lab

0.1.0 • Public • Published 2 minutes ago

Readme | Code Beta | 1 Dependency | 0 Dependents | 1 Versions

kubecon-cloudnativecon-na-2024-supply-chain-security-lab

Lab/Example - Link to the KubeCon CloudNativeCon NA 2024

Keywords

supply-chain security lab

Provenance

Built and signed on
GitHub Actions
View build summary

Source Commit	github.com/AEnguerrand/kubecon-cloudnativecon-na-20…
Build File	.github/workflows/npm.yaml
Public Ledger	Transparency log entry

Install

`> npm i @aenguerrand/kubecon-cloudnati…`

Repository

github.com/AEnguerrand/kubecon-clou…

Homepage

github.com/AEnguerrand/kubecon-clou…

Version	License
0.1.0	MIT

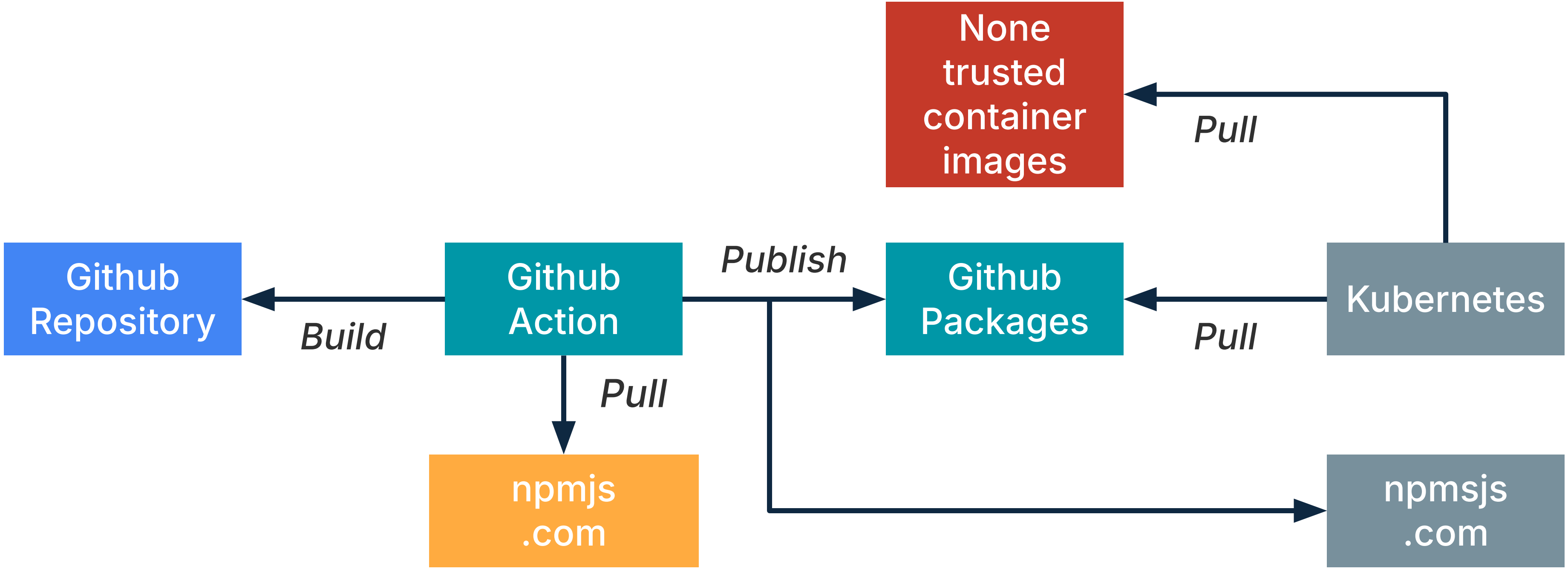
Unpacked Size	Total Files
8.18 kB	9

Issues	Pull Requests
0	0

Runtime Side: Possible defense

How to verify the application provenance running in production?

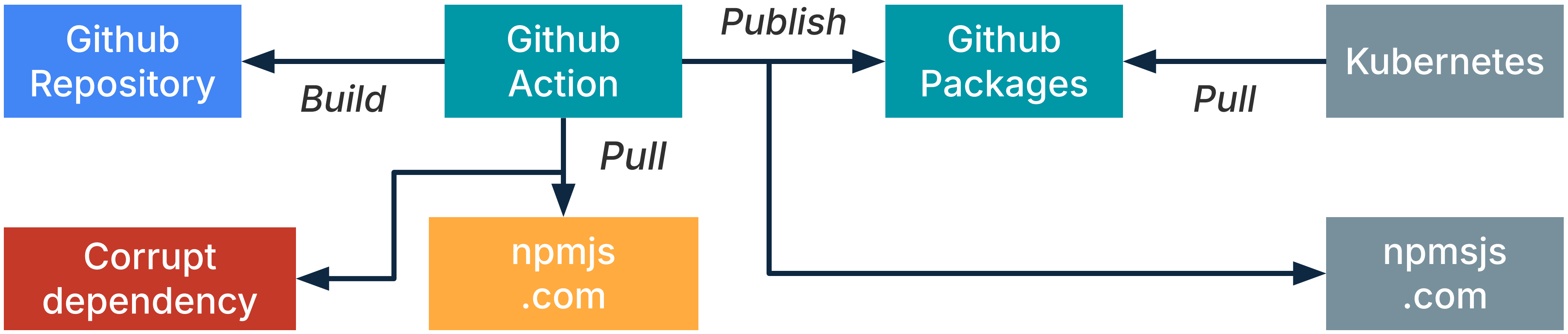
- **Kubernetes Admission Controller:** Cosign Policy Controller, Kyverno, etc.
- **Audit & Detection:** Kubescape, etc.



Runtime Side: Possible defense

How to verify the application provenance used during the build?

- **Kubernetes Admission Controller:** Cosign Policy Controller, Kyverno, etc.
- **Audit & Detection:** Kubescape, etc.



Runtime Side: Kyverno

- **Kyverno:** Open source policy engine for Kubernetes
- **Backing:** Sponsored by the CNCF
- **Purpose:** Enforces security, compliance, and operational policies in Kubernetes
- **Features:** Validates and cleans up Kubernetes resources, audits and reports policies, etc
- **Policy Format:** Policies are written as Kubernetes resources
- **Supply Chain Security Enforcement:** Ensures only properly signed and attested images are deployed

Kyverno

Website:
https://kyverno.io/

Github Organisation:
https://github.com/kyverno

Example policies link to Supply Chain Security:
https://kyverno.io/policies/?policytypes=Software%2520Supply%2520Chain%2520Security

Kyverno: Verification

- **Signature Verification:**
  Policy to check the signature of the container image

- **Attestation Verification:**
  Policy to validate the content of the attestation

- **Example:**
  Ensure that all images matching a regex pattern are signed with the correct key

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
 name: verify-image
spec:
 validationFailureAction: Enforce
 rules:
   - name: verify-image
     match:
       any:
       - resources:
           kinds:
             - Pod
     verifyImages:
     - imageReferences:
       - "ghcr.io/ledgerhq/signed-*"
       attestors:
       - entries:
         - keyless:
             subject: "https://<url-to-the-workflow>@<refs>"
             issuer: "https://token.actions.githubusercontent.com"
             rekor:
               url: https://rekor.sigstore.dev
```

Runtime Side: Kubescape

- **Kubescape:** Open source Kubernetes security platform
- **Backing:** Sponsored by CNCF (Cloud Native Computing Foundation) and linked to the Linux Foundation
- **Global Scope:** Focused on enhancing security in Kubernetes, CI/CD pipelines, and source code
- **Features:** Includes a Kubernetes scanner, CI/CD integrations, and more

Kubescape

Website:
https://kubescape.io/

Github Organisation:
https://github.com/kubescape

Kubescape: Analyse your Kubernetes Cluster

- **Scan Execution:** Run scans based on predefined controls

- **Modes:** Scans can be executed one-time or in continuous mode

- **Registry Usage:**
  - Identify usage of trusted image registries
  - Detect usage of unsafe image registries (a good first step)

- **Image Signature Verification:**
  - Check if image signatures exist (a good first step)
  - Verify image signatures for authenticity

Additional Controls:
More controls available in the documentation:
https://hub.armosec.io/docs/controls

Kubescape: Analyse your Kubernetes Cluster

Control: **C-0237**: Check if signature exists (https://hub.armosec.io/docs/c-0237)

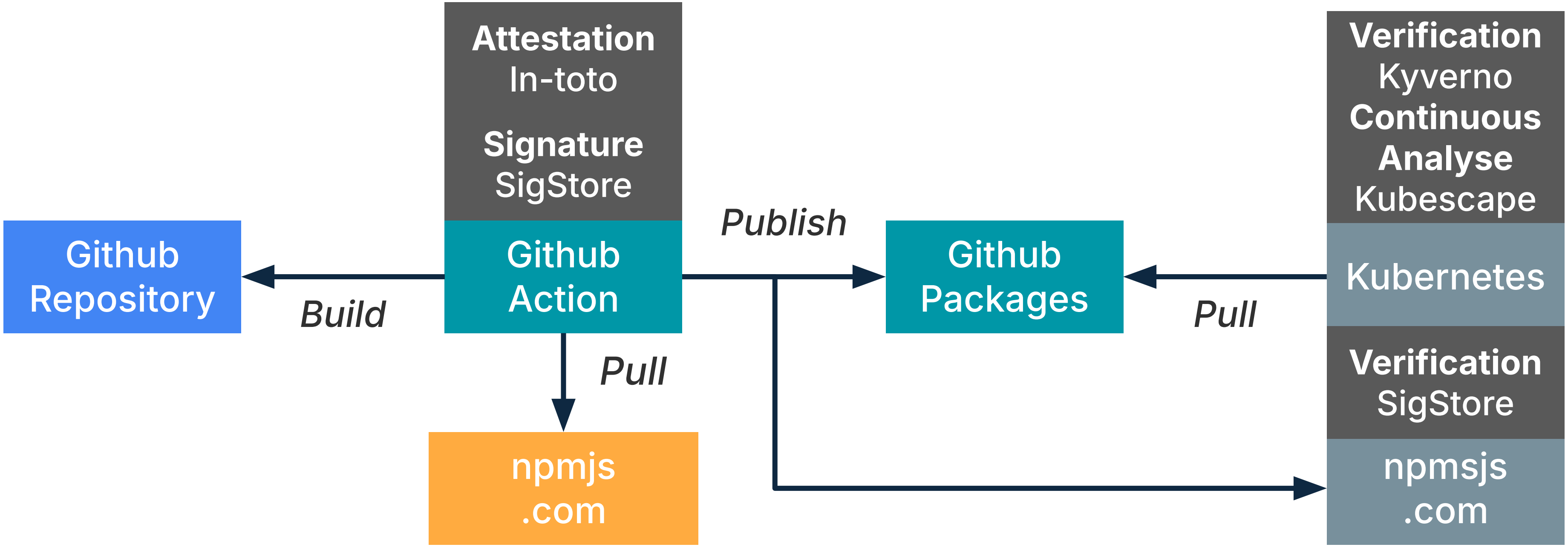
```
$ kubescape scan control "C-0237" -v
...
################################################################################
ApiVersion: apps/v1
Kind: Deployment
Name: my-hello-ledger
Namespace: default

Controls: 1 (Failed: 1, action required: 0)
```

Severity	Control name	Docs	Assisted remediation
High	Check if signature exists	https://hub.armosec.io/docs/c-0237	spec.template.spec.containers[0].image

...

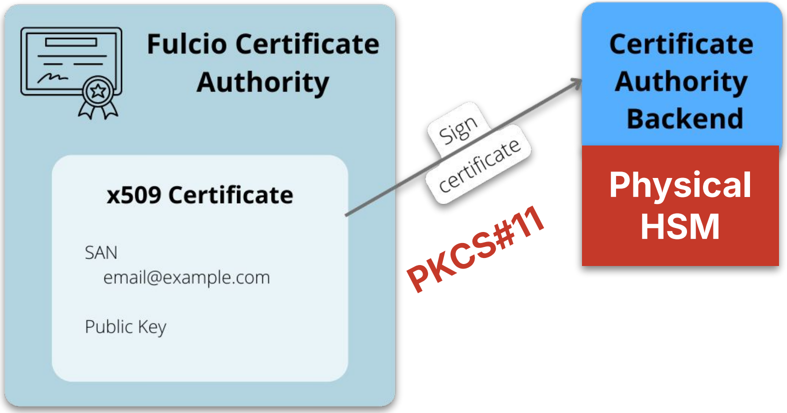
Severity	Control name	Failed resources	All Resources	Compliance score
High	Check if signature exists	8	10	20%
	Resource Summary	8	10	20.00%

Overview of the implementation

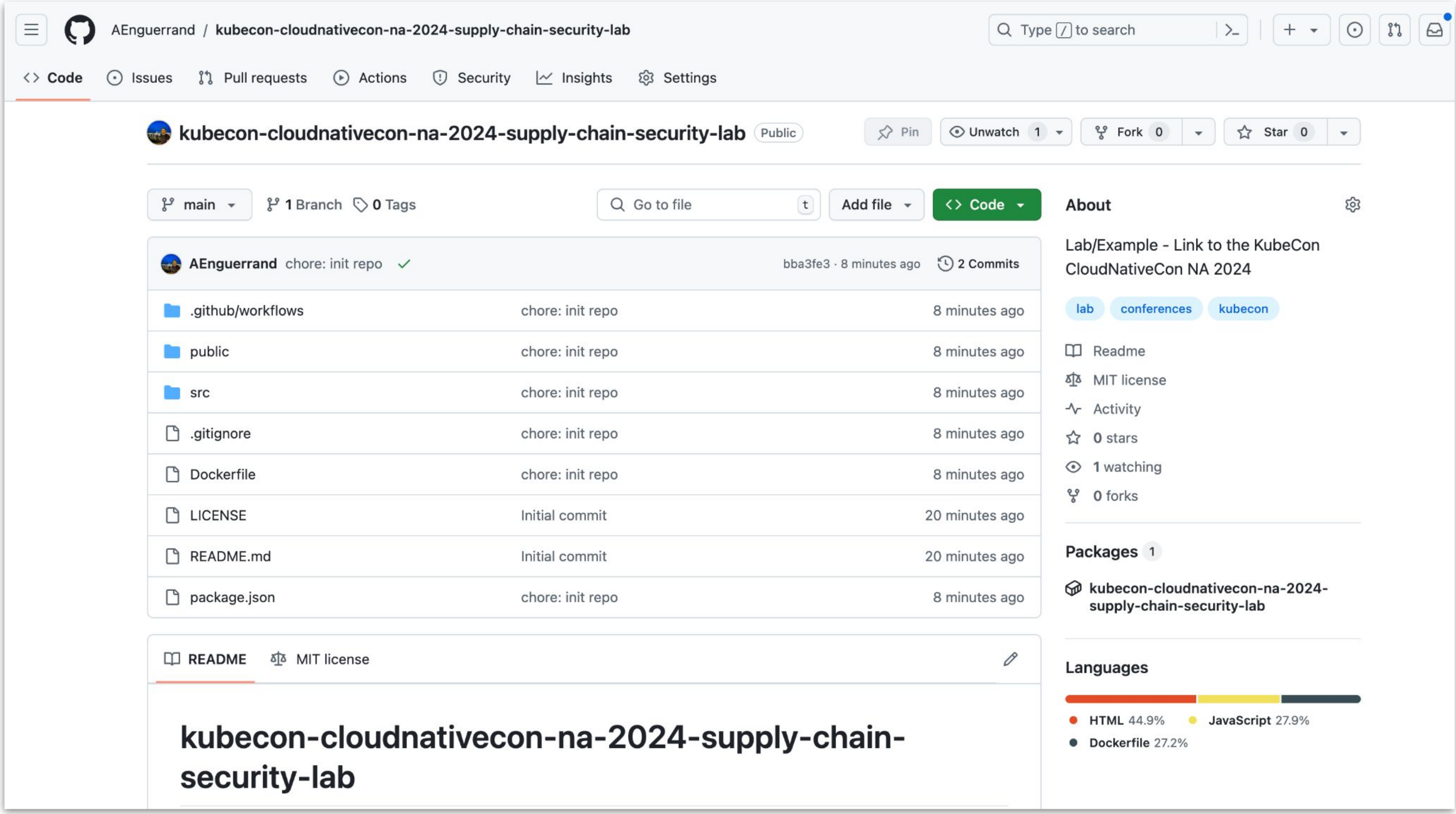


Going Further with Hardware Security Module

- **HSM** (Hardware Security Module): A physical device designed for cryptographic operations
- **Private Certificate Protection:** HSM provides a high level of physical security to protect private certificates
- **Certificate Authority (CA):** Holds the root certificate, which is used through Fulcio
- **Fulcio:** Acts as the link between your build system and the Certificate Authority
- **Sigstore Stack:** The full stack, including Fulcio and Rekor, can be hosted on Kubernetes for a self-contained solution
- **Privacy Considerations:** When signing private artifacts, using public Fulcio and Rekor services may expose information about your signature



Lab repository



Lab repository

Alternatively, use the `cosign` CLI to verify the attestation:

1. Download the package if you haven't already:

```
curl -O https://registry.npmjs.org/@aenguerrand/kubecon-cloudnativecon-na-2024-supply-chai
```

2. Retrieve the attestation from the npm registry:

```
curl https://registry.npmjs.org/-/npm/v1/attestations/@aenguerrand/kubecon-cloudnativecon-
jq '.attestations[] | select(.predicateType=="https://slsa.dev/provenance/v1").bundle' > n
```

3. Verify the attestation:

```
cosign verify-blob-attestation \
  --bundle npm-provenance.sigstore.json \
  --new-bundle-format \
  --certificate-oidc-issuer="https://token.actions.githubusercontent.com" \
  --certificate-identity="https://github.com/AEnguerrand/kubecon-cloudnativecon-na-2024-su
  kubecon-cloudnativecon-na-2024-supply-chain-security-lab-0.1.0.tgz
```

If the verification is successful, you will see:

```
Verified OK
```

Questions

- Do you have any questions or remarks?
- **Additional resources:**
  - Github repository with the example implementation: https://github.com/AEnguerrand/kubecon-cloudnativecon-na-2024-supply-chain-security-lab
  - CNCF Tag Security Whitepaper (v2): https://github.com/cncf/tag-security/blob/main/community/working-groups/supply-chain-security/supply-chain-security-paper-v2/SSCBPv2.md

Github Lab