Building Trust in Every Artifact with SBOMs

Esteban Garcia
Principal Engineer
Cloudsmith

cloudsmith

Agenda

1. Best practices for generating SBOMs for containers

2. Securely storing and indexing SBOMs alongside your artifacts

3. Validating artifacts against SBOM data before deployment

4. Using SBOMs in incident response, compliance, and auditing

What is an SBOM

“”

An SBOM, or **Software Bill of Materials**, is a detailed list of all the components, libraries, and dependencies used to build a piece of software, similar to a list of parts for a physical product. It provides transparency into the software supply chain, which is crucial for managing security, license compliance, and other risks by detailing component versions, licenses, and origins.

Esteban Garcia
Principal Engineer
Cloudsmith

“Are we running any software with X?”

Benefits of having an SBOM

01

Vulnerability Management

When a new vulnerability (like Log4j) is discovered in a specific component, you can instantly check your SBOMs to see which of your applications are affected.

This eliminates guesswork.

02

Supply Chain Transparency

SBOM provides a complete inventory of all 3rd party & OSS components within your software. Allows users to vet the components you inherit from vendors.

This builds trust with users.

03

Legal & License Compliance

OSS components come with various licenses, and failing to comply with their terms can create significant legal and financial risks.

This helps you meet regulatory standards.

An SBOM in a known format like **SPDX** or **CycloneDX** can help drive automation and trigger security alerts.

Regulatory Compliance

01

Executive Order 14028

The global adoption of the SBOM was decisively accelerated by the Executive Order 14028 in 2021, which mandated SBOMs for all federal agencies and their software vendor

02

CISA 2025 SBOM Minimum Elements

The quality and quantity of information can vary widely from one SBOM to another.

To address this, CISA, recently released a draft of its updated SBOM Minimum Elements for public comment.

03

EU Cyber Resilience Act (CRA)

The CRA is a set of rules elevating security standards for digital products in the EU. It’s another significant security regulation with a prominent SBOM requirement.

Anatomy of an SBOM

```
{
  "bomFormat": "CycloneDX",
  "specVersion": "1.5",
  "metadata": {
    "timestamp": "2025-01-15T10:30:00Z",
    "component": {
      "name": "myapp",
      "version": "1.2.3",
      "type": "container"
    }
  },
  "components": [
    {
      "type": "library",
      "name": "express",
      "version": "4.18.2",
      "purl": "pkg:npm/express@4.18.2",
      "licenses": [{"license": {"id": "MIT"}}]
    },
    {
      "type": "library",
      "name": "openssl",
      "version": "3.0.2-0ubuntu1.12",
      "purl": "pkg:deb/ubuntu/openssl@3.0.2-
0ubuntu1.12"
    }
  ],
  "dependencies": [
    {
      "ref": "pkg:npm/express@4.18.2",
      "dependsOn": ["pkg:npm/body-parser@1.20.2"]
    }
  ]
}
```

SBOM FORMAT

Describes the format of the SBOM. CycloneDX or SPDX

Anatomy of an SBOM

```
{
  "bomFormat": "CycloneDX",
  "specVersion": "1.5",
  "metadata": {
    "timestamp": "2025-01-15T10:30:00Z",
    "component": {
      "name": "myapp",
      "version": "1.2.3",
      "type": "container"
    }
  },
  "components": [
    {
      "type": "library",
      "name": "express",
      "version": "4.18.2",
      "purl": "pkg:npm/express@4.18.2",
      "licenses": [{"license": {"id": "MIT"}}]
    },
    {
      "type": "library",
      "name": "openssl",
      "version": "3.0.2-0ubuntu1.12",
      "purl": "pkg:deb/ubuntu/openssl@3.0.2-
0ubuntu1.12"
    }
  ],
  "dependencies": [
    {
      "ref": "pkg:npm/express@4.18.2",
      "dependsOn": ["pkg:npm/body-parser@1.20.2"]
    }
  ]
}
```

PROVENANCE

Who generated this SBOM?

FRESHNESS

When was this SBOM generated?

TOOLING

How was the SBOM generated?

SUBJECT

What is this SBOM describing?

Anatomy of an SBOM

```
{
  "bomFormat": "CycloneDX",
  "specVersion": "1.5",
  "metadata": {
    "timestamp": "2025-01-15T10:30:00Z",
    "component": {
      "name": "myapp",
      "version": "1.2.3",
      "type": "container"
    }
  },
  "components": [
    {
      "type": "library",
      "name": "express",
      "version": "4.18.2",
      "purl": "pkg:npm/express@4.18.2",
      "licenses": [{"license": {"id": "MIT"}}]
    },
    {
      "type": "library",
      "name": "openssl",
      "version": "3.0.2-0ubuntu1.12",
      "purl": "pkg:deb/ubuntu/openssl@3.0.2-
0ubuntu1.12"
    }
  ],
  "dependencies": [
    {
      "ref": "pkg:npm/express@4.18.2",
      "dependsOn": ["pkg:npm/body-parser@1.20.2"]
    }
  ]
}
```

IDENTITY

Component identifier
Enables precise matching during vulnerability searches

INTEGRITY

Cryptographic proof the component hasn't been tampered with
Ensures what you scanned matches what gets deployed

LICENSING

Legal terms under which the component can be used

RELATIONSHIPS

How components connect to each other

Best practices for generating SBOMs for containers

Modern CI/CD workflow using cloud-native tooling.

01

SBOM **Generation**

This should be integrated directly into the **CI pipeline**, often as a step immediately following the build or container packaging stage.

Produce a machine readable file in a standard format, such as **SPDX** or **CycloneDX**, which inventories all software components, dependencies, and their licenses.

02

SBOM **Storage**

Securely store generated SBOMs alongside the software artifact it describes. In a cloud-native workflow, this is often handled by modern artifact registries (like **Cloudsmith**) which supports OCI artifacts.

This allows the SBOM to be stored and versioned directly with its corresponding container image.

03

SBOM **Attestation**

Create cryptographic evidence that an SBOM hasn't been tampered with.

Using tools like **cosign**, sign your SBOMs and store them as attestations alongside your images.

This transforms SBOMs from static documents into verifiable claims about what went into your software

04

SBOM **Validation**

Validation acts as a quality gate within the **CD pipeline**.

Before an artifact is deployed, the stored SBOM is automatically analysed against defined security and compliance policies.

Furthermore, admission controllers like **OPA** can be used to enforce the license compliance.

SBOM Generation

Github Actions Example

```
name: Build and Generate SBOM

on: [push]

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Build image
        run: docker build -t myapp:${{ github.sha }} .

      - name: Install Trivy
        run: |
          wget -qO - https://aquasecurity.github.io/trivy-repo/deb/public.key | gpg --
dearmor | sudo tee /usr/share/keyrings/trivy.gpg > /dev/null
          echo "deb [signed-by=/usr/share/keyrings/trivy.gpg]
https://aquasecurity.github.io/trivy-repo/deb generic main" | sudo tee -a
/etc/apt/sources.list.d/trivy.list
          sudo apt-get update
          sudo apt-get install trivy

      - name: Generate SBOM
        run: |
          trivy image --format cyclonedx \
            --output sbom-${{ github.sha }}.json \
            myapp:${{ github.sha }}
```

cloudsmith

SBOM Generation

01

Generate as Part of Build, Not as Afterthought

Retrofitted SBOMs create confusion due to their time-gap leading to questions about authenticity and accuracy that cannot be definitively answered

EU Cyber Resilience Act states that "the SBOM must be generated as part of the build process or an equivalent mechanism"

02

Update SBOMs with Every Version

Dependencies change between versions, and a stale SBOM gives you a false sense of security.

NTIA requirement referenced by EU CRA and US mandate for federal software supplier

**Treat SBOMs like tests.
Required, not optional**

Modern CI/CD workflow using cloud-native tooling.

01

SBOM **Generation**

This should be integrated directly into the **CI pipeline**, often as a step immediately following the build or container packaging stage.

Produce a machine readable file in a standard format, such as **SPDX** or **CycloneDX**, which inventories all software components, dependencies, and their licenses.

02

SBOM **Storage**

Securely store generated SBOMs alongside the software artifact it describes. In a cloud-native workflow, this is often handled by modern artifact registries (like **Cloudsmith**) which supports OCI artifacts.

This allows the SBOM to be stored and versioned directly with its corresponding container image.

03

SBOM **Attestation**

Create cryptographic evidence that an SBOM hasn't been tampered with.

Using tools like **cosign**, sign your SBOMs and store them as attestations alongside your images.

This transforms SBOMs from static documents into verifiable claims about what went into your software

04

SBOM **Validation**

Validation acts as a quality gate within the **CD pipeline**.

Before an artifact is deployed, the stored SBOM is automatically analysed against defined security and compliance policies.

Furthermore, admission controllers like **OPA** can be used to enforce the license compliance.

SBOM Storage

OCI v1.1 formalised storing any artifact alongside container images

ARTIFACT TYPE

New field introduced in the spec. Explicitly identifies what an artifact is, replacing the config.mediaType workaround

SUBJECT FIELD

Subject can now be included in the manifest, allowing pointers to other OCI artifacts

REFERRERS API

New endpoint lets you discover all artifacts attached to a specific image

IMMUTABILITY

Your SBOMs will be content-addressable and cryptographically tied to the image

ORAS (OCI REGISTRY AS STORAGE)

ORAS is the CNCF tool that turns any OCI registry into an artifact store

SBOM Storage

Github Actions Example

```yaml
name: Generate SBOM

on: [push]

jobs:
  sbom:
    runs-on: ubuntu-latest
    steps:
      - name: Build image
        run: docker build -t myapp:${{ github.sha }} .

      - name: Install Trivy
        run: |
          wget -qO - https://aquasecurity.github.io/trivy-repo/deb/public.key | gpg --
dearmor | sudo tee /usr/share/keyrings/trivy.gpg > /dev/null
          echo "deb [signed-by=/usr/share/keyrings/trivy.gpg]
https://aquasecurity.github.io/trivy-repo/deb generic main" | sudo tee -a
/etc/apt/sources.list.d/trivy.list
          sudo apt-get update
          sudo apt-get install trivy

      - name: Generate SBOM
        run: |
          trivy image --format cyclonedx \
            --output sbom-${{ github.sha }}.json \
            myapp:${{ github.sha }}

      - name: Install ORAS
        run: |
          VERSION="1.1.0"
          curl -LO "https://github.com/oras-
project/oras/releases/download/v${VERSION}/oras_${VERSION}_linux_amd64.tar.gz"
          mkdir -p oras-install/
          tar -zxf oras_${VERSION}_*.tar.gz -C oras-install/
          sudo mv oras-install/oras /usr/local/bin/
          rm -rf oras_${VERSION}_*.tar.gz oras-install/

      - name: Attach SBOM to image
        run: |
          oras attach \
            --artifact-type application/vnd.cyclonedx+json \
            myapp:${{ github.sha }} \
            sbom.json:application/json
```

SBOM Storage

```
$ oras discover docker.cloudsmith.io/cloudsmith/rejekts-esteban/sbom-demo:v1.0.0 --format table

Discovered 1 artifact referencing docker.cloudsmith.io/cloudsmith/rejekts-esteban/sbom-demo:v1.0.0
Digest: sha256:87933dcc39d9b8b0c9adaf7bddfe006b059e7d18ab3d6dff37a58000a7a40bc4

Artifact Type                  Digest
application/vnd.cyclonedx+json  sha256:faa1dbf6e83b0e00f514811ea5e0acf2caa41275be8b85a4a4606560baff29c3
```

SBOM Storage

01

Co-locate SBOMs with Images

SBOM travels with the artifact automatically and it has immutable binding via content digest

02

Standardize Attachment

Use available CNCF tooling ensures it works across all compliant registries

SBOMs travel with artifacts, not behind them

Modern CI/CD workflow using cloud-native tooling.

01

SBOM **Generation**

This should be integrated directly into the **CI pipeline**, often as a step immediately following the build or container packaging stage.

Produce a machine readable file in a standard format, such as **SPDX** or **CycloneDX**, which inventories all software components, dependencies, and their licenses.

02

SBOM **Storage**

Securely store generated SBOMs alongside the software artifact it describes. In a cloud-native workflow, this is often handled by modern artifact registries (like **Cloudsmith**) which supports OCI artifacts.

This allows the SBOM to be stored and versioned directly with its corresponding container image.

03

SBOM **Attestation**

Create cryptographic evidence that an SBOM hasn't been tampered with.

Using tools like **cosign**, sign your SBOMs and store them as attestations alongside your images.

This transforms SBOMs from static documents into verifiable claims about what went into your software

04

SBOM **Validation**

Validation acts as a quality gate within the **CD pipeline**.

Before an artifact is deployed, the stored SBOM is automatically analysed against defined security and compliance policies.

Furthermore, admission controllers like **OPA** can be used to enforce the license compliance.

SBOM Attestation

In-toto is format for cryptographic attestations that link evidence to software artifacts

SIGNATURE BINDING

Attestations are cryptographically signed and linked to a specific image digest, making tampering detectable

STATEMENT ENVELOPE

In-toto wraps your SBOM in a signed statement that includes metadata about the predicate, subject, and signer

VERIFIABLE LINK

Attestations are discoverable and verifiable by anyone with access to the image digest and public key

COSIGN

CNCF tool that signs container images and attestations, storing them in OCI registries alongside your artifacts.

KEYLESS SIGNING

cosign supports keyless signing via OIDC, eliminates need to manage long-lived keys

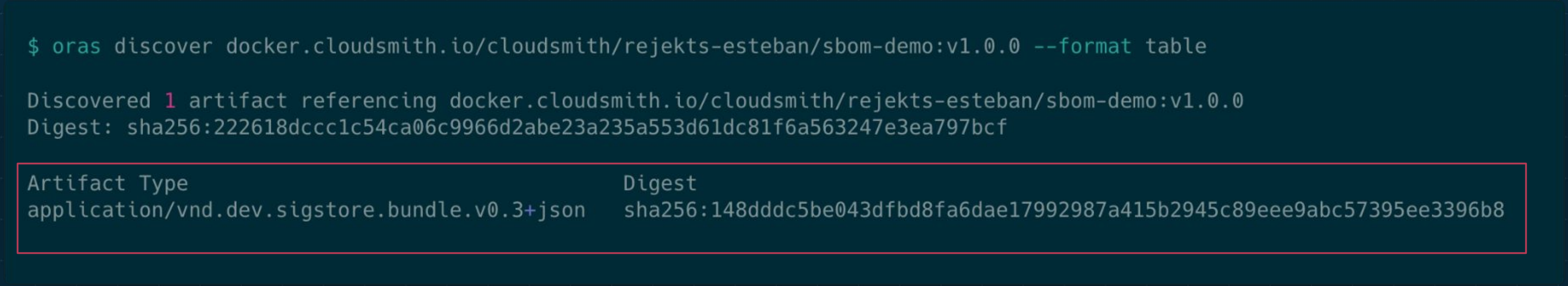
SBOM Attestation

Github Actions Example

```
name: Build and Generate SBOM

on: [push]

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Build image
        run: docker build -t myapp:${{ github.sha }} .

      - name: Install Trivy
        run: |
          wget -qO - https://aquasecurity.github.io/trivy-repo/deb/public.key | gpg --
dearmor | sudo tee /usr/share/keyrings/trivy.gpg > /dev/null
          echo "deb [signed-by=/usr/share/keyrings/trivy.gpg]
https://aquasecurity.github.io/trivy-repo/deb generic main" | sudo tee -a
/etc/apt/sources.list.d/trivy.list
          sudo apt-get update
          sudo apt-get install trivy

      - name: Generate SBOM
        run: |
          trivy image --format cyclonedx \
            --output sbom-${{ github.sha }}.json \
            myapp:${{ github.sha }}

      - name: Install cosign
        uses: sigstore/cosign-installer@v3

      - name: Attest SBOM to image
        run: |
          cosign attest --predicate sbom-${{ github.sha }}.json \
            --type cyclonedx \
            myapp:${{ github.sha }}
        env:
          COSIGN_EXPERIMENTAL: 1
```

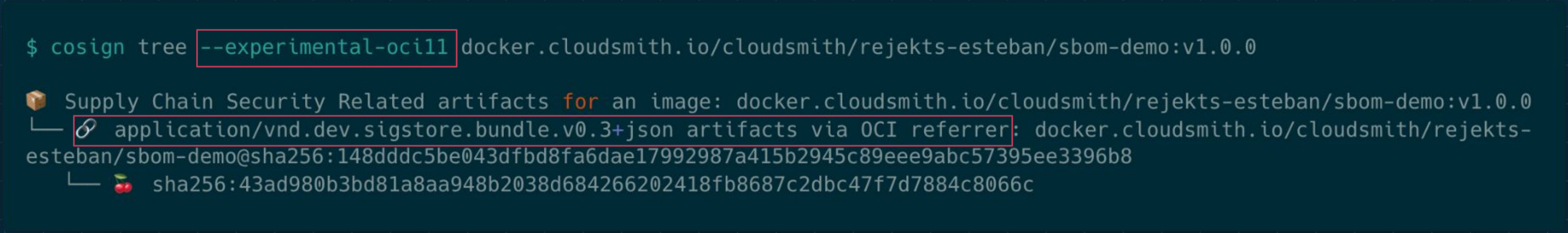
cloudsmith

SBOM Attestation



```
$ oras discover docker.cloudsmith.io/cloudsmith/rejekts-esteban/sbom-demo:v1.0.0 --format table

Discovered 1 artifact referencing docker.cloudsmith.io/cloudsmith/rejekts-esteban/sbom-demo:v1.0.0
Digest: sha256:222618dccc1c54ca06c9966d2abe23a235a553d61dc81f6a563247e3ea797bcf

Artifact Type                                 Digest
application/vnd.dev.sigstore.bundle.v0.3+json   sha256:148dddc5be043dfbd8fa6dae17992987a415b2945c89eee9abc57395ee3396b8
```

SBOM Attestation



```
$ cosign tree --experimental-oci11 docker.cloudsmith.io/cloudsmith/rejekts-esteban/sbom-demo:v1.0.0

📦 Supply Chain Security Related artifacts for an image: docker.cloudsmith.io/cloudsmith/rejekts-esteban/sbom-demo:v1.0.0
└── 🔗 application/vnd.dev.sigstore.bundle.v0.3+json artifacts via OCI referrer: docker.cloudsmith.io/cloudsmith/rejekts-
esteban/sbom-demo@sha256:148dddc5be043dfbd8fa6dae17992987a415b2945c89eee9abc57395ee3396b8
   └── 🍒 sha256:43ad980b3bd81a8aa948b2038d684266202418fb8687c2dbc47f7d7884c8066c
```

SBOM Attestation

01

Cryptographic proof

Attestation transforms your SBOM from a static document into verifiable evidence. The signature proves the SBOM hasn't been tampered with and ties it to a specific artifact digest

02

Sign at Build Time

Create attestations immediately after generating your SBOM in CI/CD. This creates an immutable record of what went into your software at a known point in time

03

Enable Supply Chain Provenance

Attestations create a verifiable chain linking the SBOM to the builder, signer identity, and timestamp. This satisfies compliance requirements (SLSA, EU CRA) that demand proof of software origin and integrity

Attestations turn SBOMs from claims into cryptographic proof

Modern CI/CD workflow using cloud-native tooling.

01

SBOM **Generation**

This should be integrated directly into the **CI pipeline**, often as a step immediately following the build or container packaging stage.

Produce a machine readable file in a standard format, such as **SPDX** or **CycloneDX**, which inventories all software components, dependencies, and their licenses.

02

SBOM **Storage**

Securely store generated SBOMs alongside the software artifact it describes. In a cloud-native workflow, this is often handled by modern artifact registries (like **Cloudsmith**) which supports OCI artifacts.

This allows the SBOM to be stored and versioned directly with its corresponding container image.

03

SBOM **Attestation**

Create cryptographic evidence that an SBOM hasn't been tampered with.

Using tools like **cosign**, sign your SBOMs and store them as attestations alongside your images.

This transforms SBOMs from static documents into verifiable claims about what went into your software

04

SBOM **Validation**

Validation acts as a quality gate within the **CD pipeline**.

Before an artifact is deployed, the stored SBOM is automatically analysed against defined security and compliance policies.

Furthermore, admission controllers like **OPA** can be used to enforce the license compliance.

SBOM Validation

Attestations prove who created the SBOM. Validation policies prove what's in it

DEPLOYMENT TIME

Validate SBOM attestations at the Kubernetes admission controller level before images reach production.

POLICY ENFORCEMENT

Some registries allow for policy enforcement to reject images without valid SBOMs

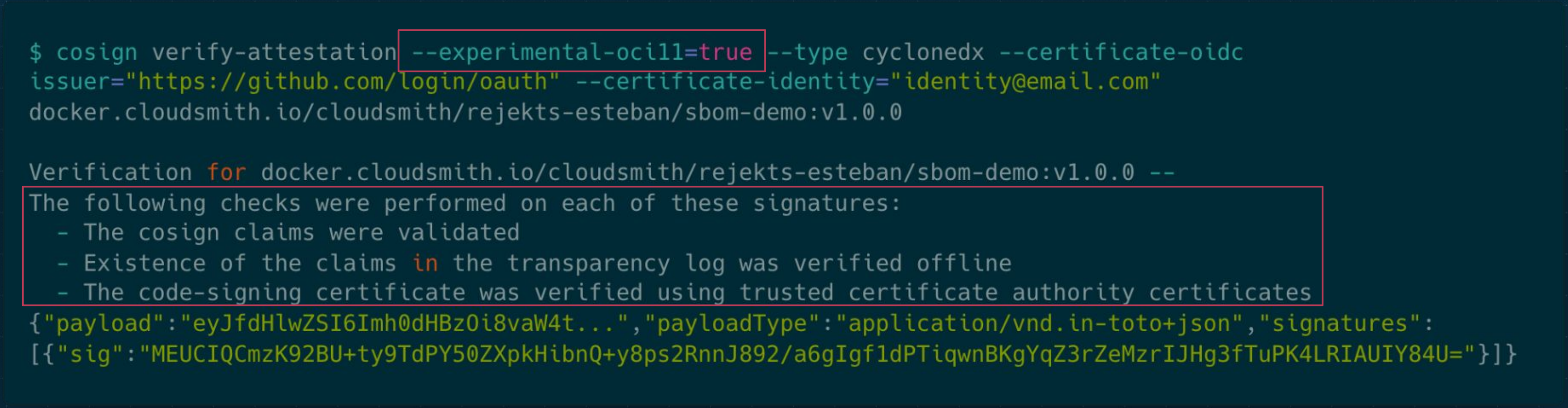
INCIDENT RESPONSE

When vulnerabilities are discovered, validate the SBOM to confirm exactly what components were in deployed images

TRIVY

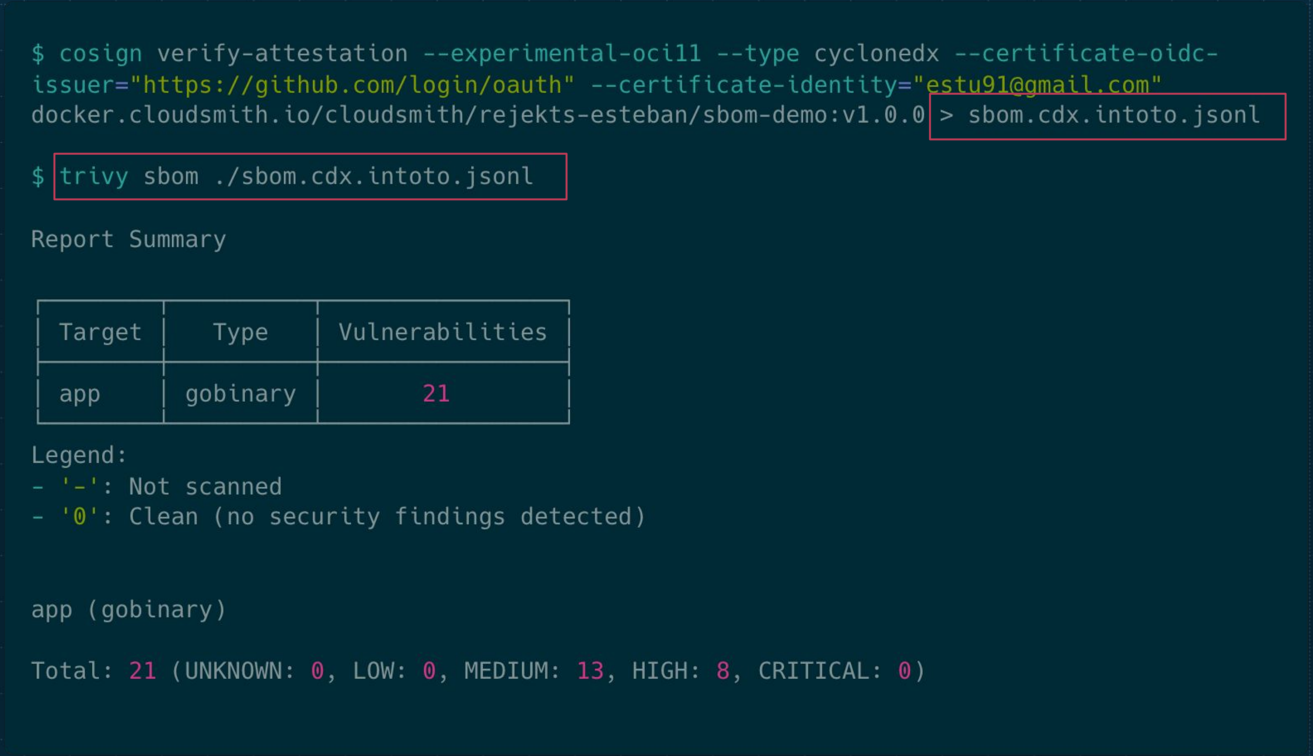
You can lean on cosign for attestation verification and on trivy for vulnerability scanning on the SBOM

SBOM Validation



```
$ cosign verify-attestation --experimental-oci11=true --type cyclonedx --certificate-oidc
issuer="https://github.com/login/oauth" --certificate-identity="identity@email.com"
docker.cloudsmith.io/cloudsmith/rejekts-esteban/sbom-demo:v1.0.0

Verification for docker.cloudsmith.io/cloudsmith/rejekts-esteban/sbom-demo:v1.0.0 --
The following checks were performed on each of these signatures:
  - The cosign claims were validated
  - Existence of the claims in the transparency log was verified offline
  - The code-signing certificate was verified using trusted certificate authority certificates
{"payload":"eyJfdHlwZSI6Imh0dHBzOi8vaW4t...","payloadType":"application/vnd.in-toto+json","signatures":
[{"sig":"MEUCIQCmzK92BU+ty9TdPY50ZXpkHibnQ+y8ps2RnnJ892/a6gIgf1dPTiqwnBKgYqZ3rZeMzrIJHg3fTuPK4LRIAUIY84U="}]}
```

SBOM Validation



```
$ cosign verify-attestation --experimental-oci11 --type cyclonedx --certificate-oidc-
issuer="https://github.com/login/oauth" --certificate-identity="estu91@gmail.com"
docker.cloudsmith.io/cloudsmith/rejekts-esteban/sbom-demo:v1.0.0 > sbom.cdx.intoto.jsonl

$ trivy sbom ./sbom.cdx.intoto.jsonl

Report Summary

┌────────┬───────────┬─────────────────┐
│ Target │   Type    │ Vulnerabilities │
├────────┼───────────┼─────────────────┤
│ app    │ gobinary  │       21        │
└────────┴───────────┴─────────────────┘
Legend:
- '-': Not scanned
- '0': Clean (no security findings detected)


app (gobinary)

Total: 21 (UNKNOWN: 0, LOW: 0, MEDIUM: 13, HIGH: 8, CRITICAL: 0)
```

cloudsmith

SBOM Validation in Kubernetes

```yaml
apiVersion: policy.sigstore.dev/v1alpha1
kind: ClusterImagePolicy
metadata:
  name: sbom-attestation-policy
spec:
  images:
    - glob: "docker.cloudsmith.io/**"
  authorities:
    - name: keyless
      keyless:
        url: https://fulcio.sigstore.dev
        identities:
          - issuer: https://github.com/login/oauth
            subject: "issuer@gmail.com"
      ctlog:
        url: https://rekor.sigstore.dev
      attestations:
        - name: sbom-check
          predicateType: https://cyclonedx.org/bom
```

SBOM Validation in Kubernetes

OCI v1.1 attached attestations aren’t supported yet

SBOM Validation in Kubernetes

```yaml
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sSBOMValidation
metadata:
  name: block-vulnerable-packages
spec:
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Pod"]
      - apiGroups: ["apps"]
        kinds: ["Deployment", "StatefulSet", "DaemonSet"]
  parameters:
    provider: sbom-provider
    certIdentity: "issuer@gmail.com"
    certOidcIssuer: "https://github.com/login/oauth"
    prohibitedPackages:
      - name: "log4j-core"
        version: "2.14.1"  # Vulnerable version
      - name: "spring-core"
        version: "5.2.0"   # Vulnerable version
    prohibitedLicenses:
      - "GPL-3.0"
      - "AGPL-3.0"
```

SBOM Validation in Kubernetes

```
package k8ssbomvalidation

violation[{"msg": msg}] {
  # Get container images
  container := input_containers[_]
  image := container.image

  # Build key with image and imagePullSecrets
  key := build_key(image)

  # Query SBOM from external provider
  provider := object.get(input.parameters, "provider", "sbom-provider")
  response := external_data({"provider": provider, "keys": [key]})

  # Check if the response is empty or the key is missing
  count(response) == 0
  msg := sprintf("Failed to verify attestation or retrieve SBOM for image: %v", [image])
 }
```

SBOM Validation in Kubernetes

This is just a proof of concept on how to use opa-gatekeepers new external data provider

How does Cloudsmith achieve this?

Cloudsmith uses these open-source projects to automate the generation of SBOMs

TRIVY

By leveraging Trivy's SBOM capabilities, we ensure that every image has a clear and standardised inventory

CYCLONEDX

Every Docker/OCI image published to Cloudsmith is automatically accompanied by an SBOM in the CycloneDX format.

TRIVY

There are a number of tools to generate an SBOM, including Trivy, Syft, Docker Scout, BuildKit Cloudsmith supports the creation of SBOM of Docker image with Trivy.

ENTERPRISE POLICY MANAGER

Our Policy Manager uses OPA behind-scenes to allow/deny the download of packages using REGO

COSIGN

All Container Images are automatically signed with cosign on push

SBOM Validation with Cloudsmith's EPM

```
default match := false

permissive_licenses := {"MIT", "ISC", "BSD-2-Clause", "BSD-3-Clause"}

match if {
    input.v0.sbom != null
    component := input.v0.sbom.components[_]
    component.licenses[_].license.id in permissive_licenses
}
```

Learn more about it here

SBOM Validation

01

Attestation Verification ≠ SBOM Validation

Verifying the attestation signature proves authenticity and prevents tampering. Validating the SBOM contents checks policies — vulnerabilities, licenses, components. You need both

02

Validation Must Happen at Admission

Validate at deployment time, not after. If you validate after deployment, you're already running untrusted code

03

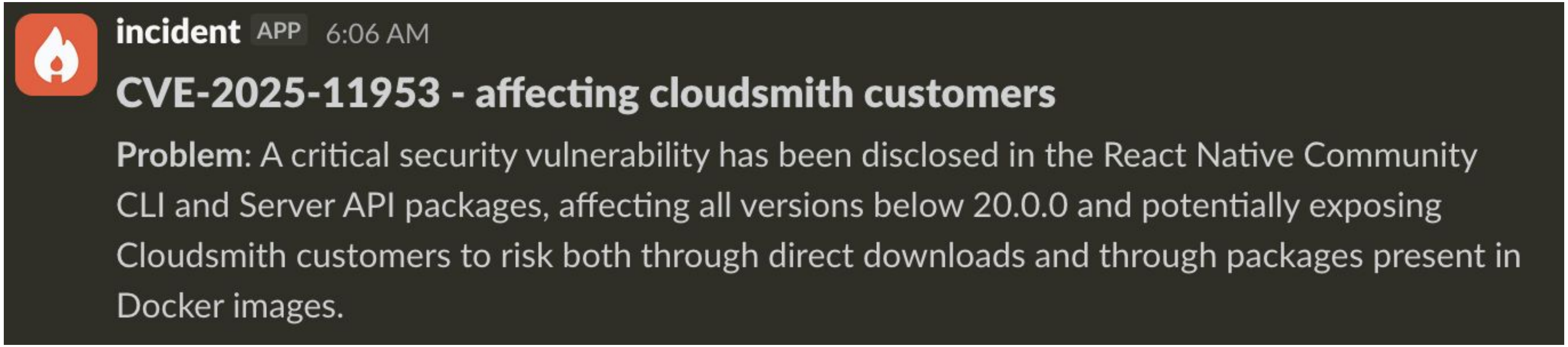
Tooling Ecosystem Matters

Different tools handle different aspects — cosign verifies, Trivy analyzes, OPA/Gatekeeper enforces policies. Pick the right tool for each job. A chain is only as strong as its weakest link

Signed doesn't mean safe. Validated does

Using SBOMs in incident response, compliance, & auditing

CVE-2025-11953 - Real-life Incident Example



incident APP 6:06 AM

CVE-2025-11953 - affecting cloudsmith customers

Problem: A critical security vulnerability has been disclosed in the React Native Community CLI and Server API packages, affecting all versions below 20.0.0 and potentially exposing Cloudsmith customers to risk both through direct downloads and through packages present in Docker images.

CVE-2025-11953 - Real-life Incident Example

01

Check all SBOMs for the specific package versions

Because we have SBOMs for all Docker Images in Cloudsmith we can check all of them simultaneously

02

Found affected customers

We found 1200+ affected packages across 13 customers

03

Notify and mitigate

We notified all customers of the vulnerability and recommended policies that can be applied

From declared incident to mitigation <1hr

Supporting Resources

https://github.com/estebangarcia/rejekts-atlanta-sbom

Thank you