Securing Your Software Supply Chain

Meet The Instructor - Marc Boorshtein

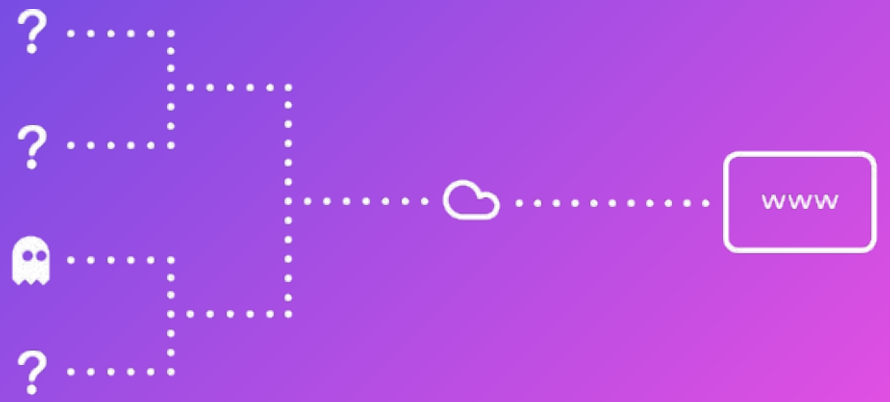
- CTO Tremolo Security
- Identity Management expert for 20+ years
- Experience in commercial and federal agencies
- Kubernetes since 2015
- Co-Author Kubernetes an Enterprise Guide: 2nd Ed
    - **Amazon 20% Discount - 20KAEG**

Meet The Instructor - John Osborne

-

sigstore

Making sure your software's what it claims to be

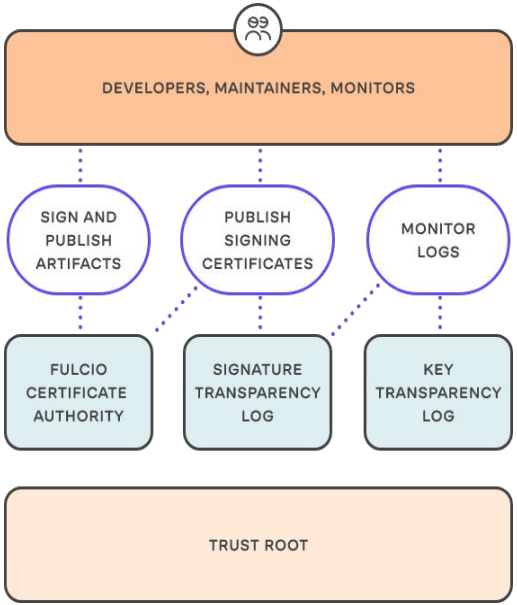


A new standard for signing, verifying and protecting software.

Modern **'keyless'** signing removes painful key management.

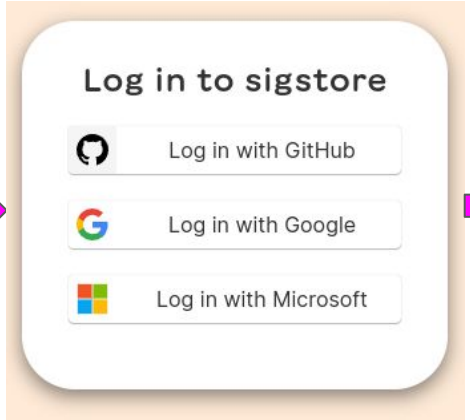
Sigstore, a new standard for signing, verifying and protecting software

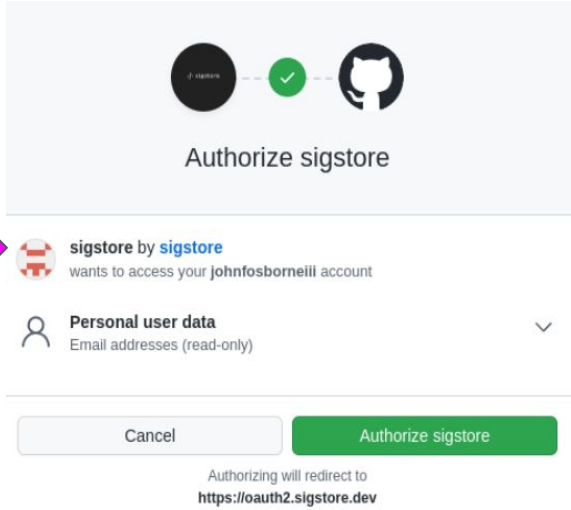
- All cryptographically verifiable, auditable, community operated
- Key Pieces:
  - Cosign: CLI to sign and verify artifacts
  - Fulcio: CA that issues *free* short-lived code signing certs
  - Rekor: transparency log for signatures and metadata
- *Sigstore community hosts a free shared public-good instance that you can use right now*



Key Automation based on identity (or KMS)

```
josborne@fedora-system76:~
❯ cosign sign-blob frontend-deployment.yaml
Using payload from: frontend-deployment.yaml
Generating ephemeral keys...
Retrieving signed certificate...
```

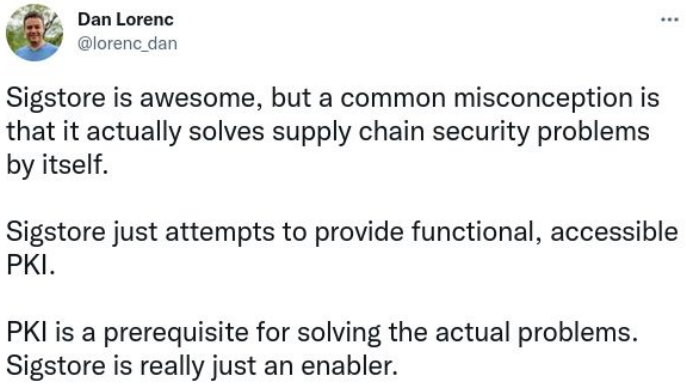




@mlbiam / @CloudLvlMidnite

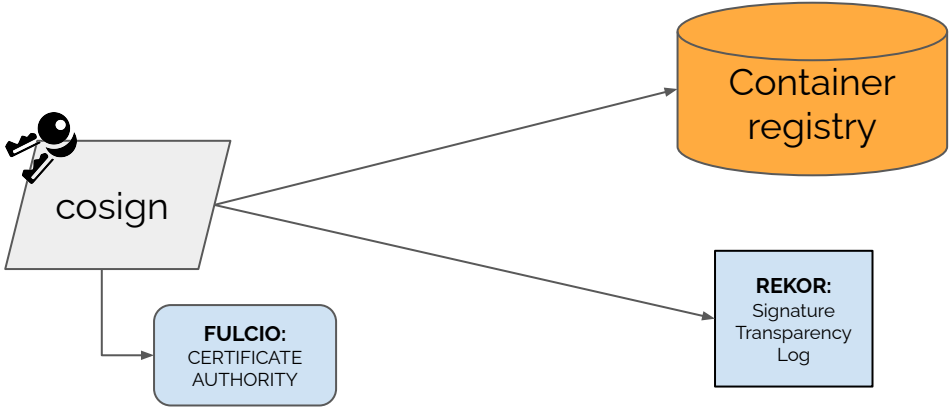
What do signatures guarantee?

- Signatures give you evidence that:
  - What you signed hasn’t changed
  - It came from the producer that signed it
- We want to sign our software
  - Artifacts (.jars, container images)
  - Git Commits
- We want to sign our supporting docs
  - SBOMs
  - Attestations



Signing With Cosign

1. Obtains keypair (locally, keyless, or KMS)
2. Requests code signing certificate
3. Downloads container manifest & sign
4. Uploads signature, public key (and certificate chain) to registry as OCI image
5. Creates entry in rekor




```
# Let's Encrypt
X509v3 Key Usage: critical
    Digital Signature, Key Encipherment
X509v3 Extended Key Usage:
    TLS Web Server Authentication

# Fulcio
X509v3 Key Usage: critical
    Digital Signature
X509v3 Extended Key Usage:
    Code Signing
```

Signing and Verifying

Images	$ cosign sign josborne/myimage
Blobs	$ cosign sign-blob ./myblob
Attestations	$ cosign attest josborne/myimage --predicate ./att.json
SBOMs	$ cosign attest josborne/myimage --predicate ./sbom

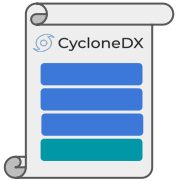
Images	$ cosign verify josborne/myimage
Blobs	$ cosign verify-blob ./myblob
Attestations	$ cosign verify-attestation josborne/myimage
SBOMs	$ cosign verify-attestation josborne/myimage

What should we be signing?

Code

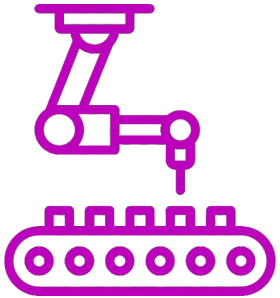
Compiled Artifacts

Attestations



Common Attestations

How it was Built (Provenance)

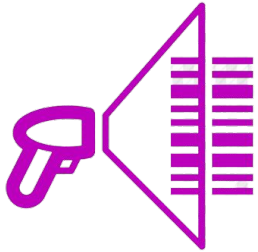


ko attests to the fact that it built a container image with digest "sha256:87f7fe…" from git commit "f0c93d…"

How it was Tested

GitHub Actions attests to the fact that the npm tests passed on git commit "f0c93d…".

What Security Scans it Passed



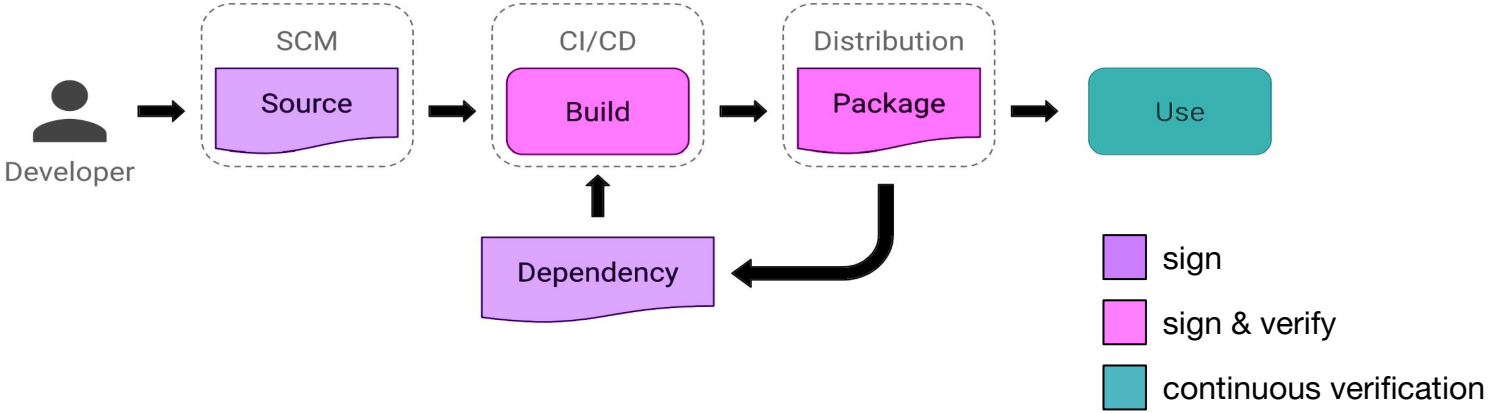
GitHub Actions attests to the fact that no vulnerabilities were found in container image "sha256:87f7fe…" at a particular time using a scanning tool

Supply Chain Guidance - Sign all interactions

SLSA

NIST SSDF

CIS Benchmarks



@mlbiam / @CloudLvlMidnite

SLSA Level 1

SLSA Level 1

- Scripted Builds
- Provenance Available

```yaml
_type: https://in-toto.io/Statement/v0.1
predicateType: https://slsa.dev/provenance/v0.2
predicate:
  buildConfig:
    steps:
      - entryPoint: /usr/bin/mvn
        arguments:
          - '-s'
          - settings.xml
          - $(params.GOALS)
          - '-Dmaven.test.skip'
      - entryPoint:
          #!/usr/bin/env bash
          cat > Dockerfile.gen <<EOF
          FROM registry.access.redhat.com/ubi8/openjdk-17:1.12
          COPY --chown=185 target/* /deployments/

          EXPOSE 8080

          ENV JAVA_OPTS="-Dquarkus.http.host=0.0.0.0 -Djava.util.logging.manager=org.jboss.logmanager.LogManager"
          ENV JAVA_APP_JAR="/deployments/quarkus-run.jar"
          EOF
```

SLSA Level 1 Attestation Example

SLSA Level 2

SLSA Level 2

- Use a Build Service
- Sign & Verify Provenance

```
cosign attest --type slsaprovenance
```

Sign


```
_type: https://in-toto.io/Statement/v0.1
predicateType: https://slsa.dev/provenance/v0.2
subject:
  - name: ghcr.io/metal-toolbox/audittail
predicate:
  builder:
    id: https://github.com/metal-toolbox/auditevent/Attestations/GitH
  buildType: https://github.com/Attestations/GitHubActionsWorkflow@v1
  invocation:
    configSource:
      uri: git+https://github.com/metal-toolbox/auditevent
      digest:
        sha1: a8cf6a8ec35706b07249c1ff38d06122c08173b5
  metadata:
    buildInvocationID: https://github.com/metal-toolbox/auditevent/ac
  materials:
    - uri: git+https://github.com/metal-toolbox/auditevent
      digest:
        sha1: a8cf6a8ec35706b07249c1ff38d06122c08173b5
```


Verify Signature

```
cosign verify-attestation
```

SLSA Level 2 Attestation Example

@mlbiam / @CloudLvlMidnite

SLSA Level 3

SLSA Level 3

- Build As Code
- Non-Falsifiable Provenance
- External KMS

```
cosign attest --type slsaprovenance
```

Sign


```
_type: https://in-toto.io/Statement/v0.1
predicateType: https://slsa.dev/provenance/v0.2
predicate:
  builder:
    id: https://github.com/laurentsimon/slsa-github-generator-ko/.github/workflows/slsa3-builder.yml@
  buildType: https://github.com/slsa-framework/slsa-github-generator-go@v1
  invocation:
    configSource:
      uri: git+https://github.comlaurentsimon/slsa-on-github-test@refs/heads/main.git
      digest:
        sha1: ad1ada158145ccfa006aac936061d0300468542f
      entryPoint: Ko Caller
  buildConfig:
    steps:
      - command:
          - ~/go/bin/ko
          - publish
          - '-B'
          - '--platform=linux/amd64,linux/arm64,linux/386,linux/arm'
          - '--tags=tag5,tag6'
          - laurentsimon/helloworld
        env:
          - KO_DOCKER_REPO=laurentsimon/helloworld
  materials:
    - uri: git+laurentsimon/slsa-on-github-test.git
      digest:
        sha1: ad1ada158145ccfa006aac936061d0300468542f
```


Verify Attestation

```
cosign verify-attestation
```

SLSA Level 3 Attestation Example

@mlbiam / @CloudLvlMidnite

SLSA Level 4 Code Review

```
cosign attest --type slsaprovenance
```

Dan Signs

```
cosign attest --type slsaprovenance
```

Kim Signs

```
_type: https://in-toto.io/Statement/v0.1
subject:
- digest:
    sha1: 859b387b985ea0f414e4e8099c9f874acb217b94
predicateType: https://chainguard.dev/CodeReview/v1
predicate:
  repo:
    uri: https://github.com/example/my-project
    branch: main
  author: mailto:engineering@chainguard.dev
  reviewers:
  - mailto:dan@chainguard.dev
  - mailto:kim@chainguard.dev
```

Verify Dan

```
cosign verify-attestation
```

Verify Kim

```
cosign verify-attestation
```

@mlbiam / @CloudLvlMidnite

gitsign

"Keyless" git commit signing with Sigstore

- Sign with git commit -S
- Sign every commit by default (optional)

sigstore
gitsign

Meet SLSA Verified-History
& Two-Person Review Reqs
github.com/sigstore/gitsign

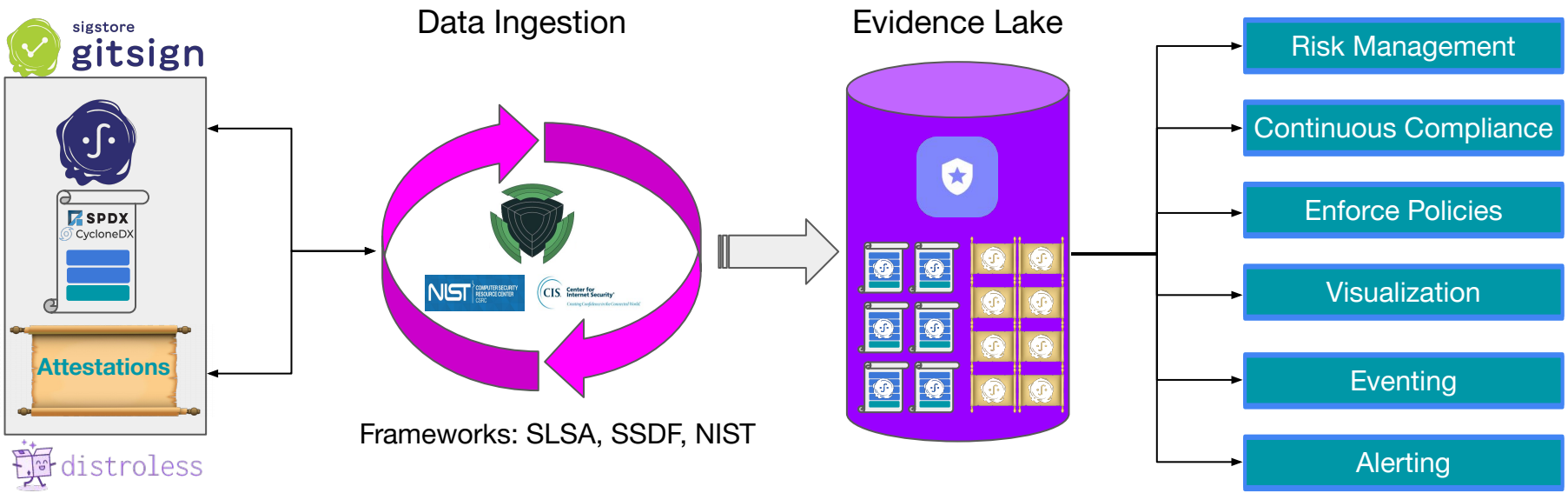
Certificate subject

Certificate issuer

CN	sigstore
O	sigstore.dev

Learn about vigilant mode.

Making the Information Actionable

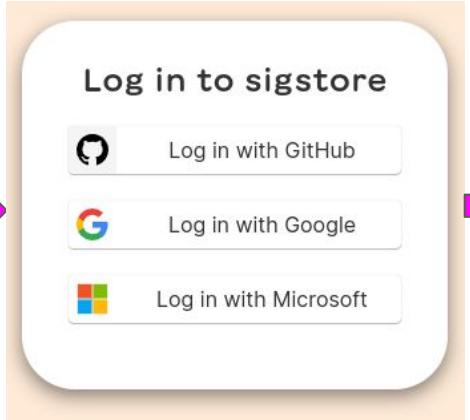


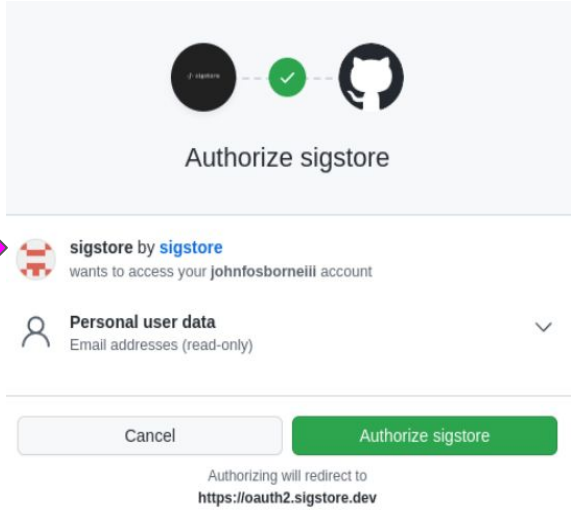
@mlbiam / @CloudLvlMidnite

Back in slide 4…Key Automation based on identity (or KMS)

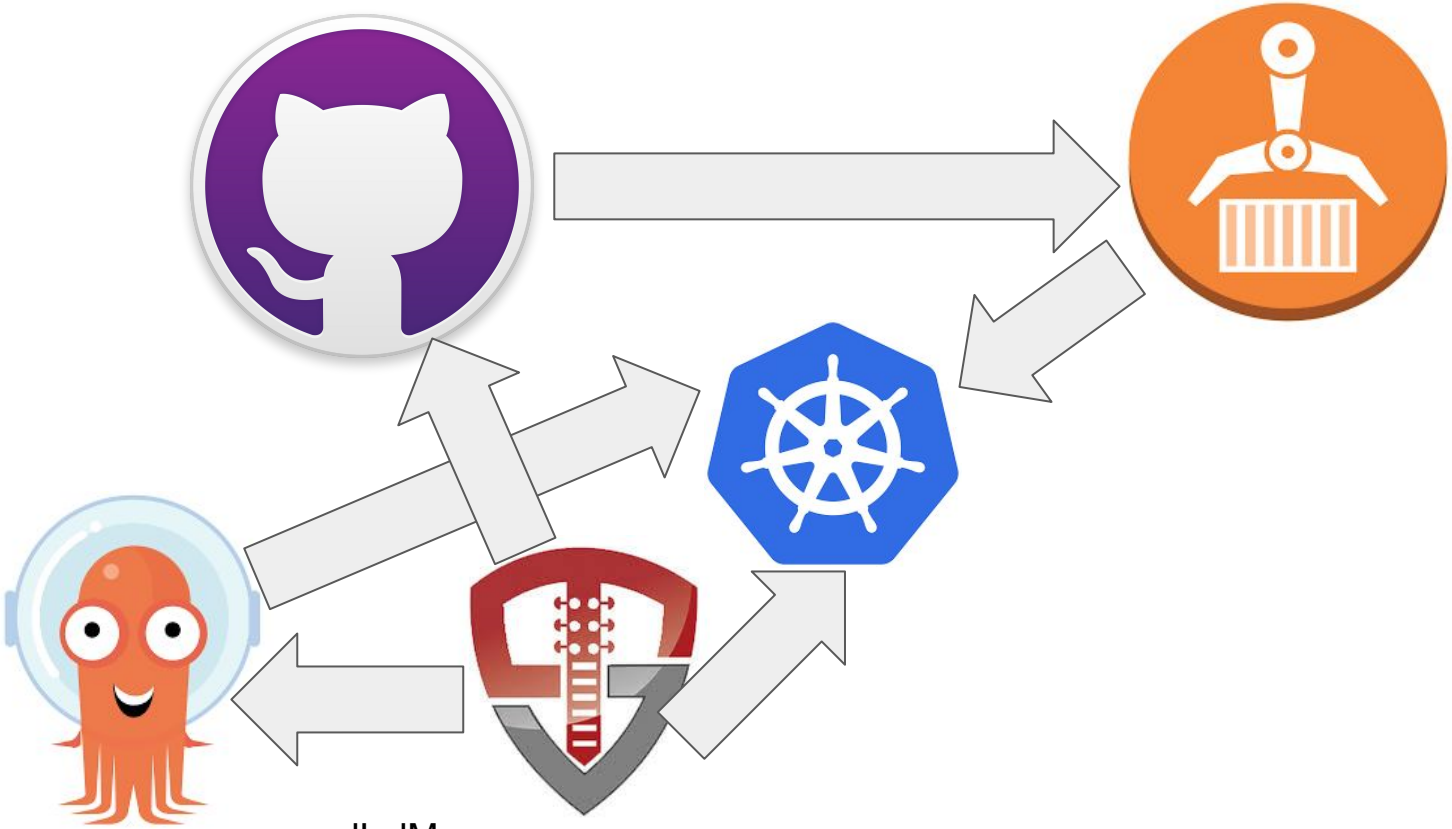
It’s all about the identity, not the key

```
josborne@fedora-system76:~
❯ cosign sign-blob frontend-deployment.yaml
Using payload from: frontend-deployment.yaml
Generating ephemeral keys...
Retrieving signed certificate...
```





Test Lab Build Infrastructure



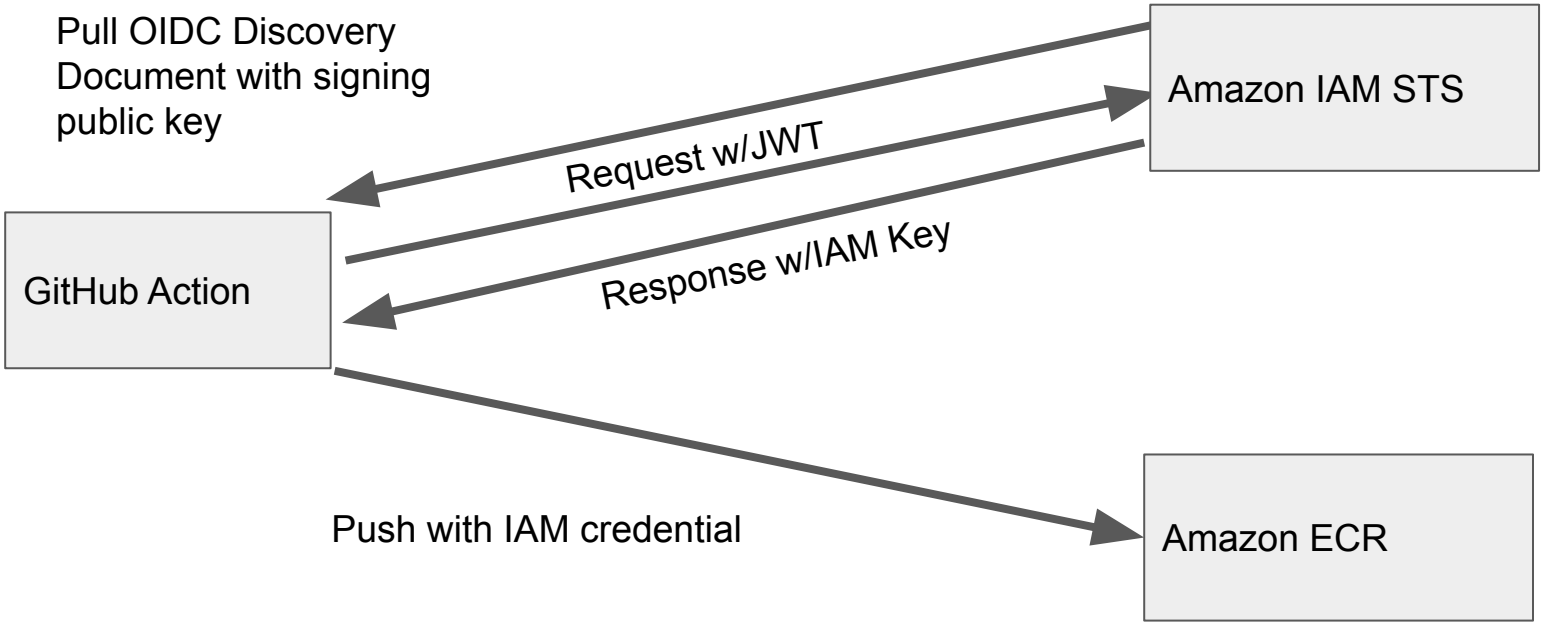
Securing your build infrastructure

- Authenticating requests between components
- OpenID Connect
- What’s in a JWT?
- How to verify a JWT
- Risks of bearer tokens
- Lab - Building, signing, & verifying an app container
- Anti patterns - static tokens, passwords, maybe x509 certs
- Making attestations to your build provenance

Authenticating requests between components

- Can you **trust** a build?
- Goals
  - Short lived tokens
  - Leverage existing identity
  - Easily rotate identity information
  - Leverage Public Key Infrastructure
- What to avoid
  - Static tokens
  - Personal access tokens
  - Passwords

OpenID Connect in Infrastructure



What’s In a JWT?

Section	Example	Decoded

Header	eyJraWQiOiJDPU15Q291bnRyeSwgU1Q9U3RhdGUgb2YgQ2x1c3RlciwgTD1NeSBDbHVzdGVyLCBPPU15T3JnLCBPVT1LdWJlcm5ldGVzLCBDTj11bmlzb24tc2FtbDItcnAtc2lnLUM9TXlDb3VudHJ5LCBTVD1TdGF0ZSBvZiBDbHVzdGVyLCBMPU15IENsdXN0ZXIsIE89TXlPcmcsIE9VPUt1YmVybmV0ZXMsIENOPXVuaXNvbi1zYW1sMi1ycC1zaWctMTY2MTE5NDg1NjY0OCIsImFsZyI6IlJTMjU2In0	{
		"kid": "C=MyCountry, ST=State of Cluster, L=My Cluster, O=MyOrg, OU=Kubernetes, CN=unison-saml2-rp-sig-C=MyCountry, ST=State of Cluster, L=My Cluster, O=MyOrg, OU=Kubernetes, CN=unison-saml2-rp-sig-1661194856648",
		"alg": "RS256"
		}

Body	eyJpc3MiOiJodHRwczovL2s4c291LmFwcHMuMTkyLTE2OC0yLTgubmlwLmlvL2F1dGgvaWRwL2s4c0lkcCIsImF1ZCI6Imt1YmVybmV0ZXMiLCJleHAiOjE2NjIyMzkxOTAsImp0aSI6IlZqQVliTnhMVE43bGVSWEdHUE13U3ciLCJpYXQiOjE2NjIyMzkxMzAsIm5iZiI6MTY2MjIzOTAxMCwic3ViIjoiamphY2tzb24iLCJuYW1lIjoiIEphY2tzb24iLCJncm91cHMiOlsidXNlcnMiLCJrOHMtY2x1c3RlciI1rOHMtYWRtaW5pc3RyYXRvcnMtaW50ZXJuYWwiXSwicHJlZmVycmVkX3VzZXJuYW1lIjoiamphY2tzb25AdHJlbW9sb3NlY3VyaXR5LmNvbSJ9	{
		"iss": "https://k8sou.apps.192-168-2-8.nip.io/auth/idp/k8sIdp",
		"aud": "kubernetes",
		"exp": 1662239190,
		"jti": "VjAYbNxLTN7leRXGGPMwSw",
		"iat": 1662239130,
		"nbf": 1662239010,
		"sub": "jjackson",
		"name": " Jackson",
		"groups": [
		"users",
		"k8s-cluster-k8s-administrators-internal"
		],
		"preferred_username": "jjackson",
		"email": "marc+jjackson@tremolosecurity.com"
		}

JWT Body


```
{
  "iss": "https://k8sou.apps.192-168-2-8.nip.io/auth/idp/k8sIdp",
  "aud": "kubernetes",
  "exp": 1662239190,
  "jti": "VjAYbNxLTN7leRXGGPMwSw",
  "iat": 1662239130,
  "nbf": 1662239010,
  "sub": "jjackson",
  "name": " Jackson",
  "groups": [
    "users",
    "k8s-cluster-k8s-administrators-internal"
  ],
  "preferred_username": "jjackson",
  "email": "marc+jjackson@tremolosecurity.com"
}
```

What’s In a JWT?

Section	Example	Decoded
Signature	acuJ-Z-IWMXx2Wuf8ptFbkLuFvZN4M-0XcgJUo5vyf4KqJzdLn5NciIWmHdAE0xqQPsomwoimAT3ipZUcSieJoo9dLwQg-Sm__c22f6l0_Rft0fAYtHqxHEGLgfeXhFtf7VROV5FVpUqVJMzhVJ3vJzBma3CAZo6-kSkgSks1bycu4bBeRrJKFPE8QvwFkw5zwB29OW5BWYs2eZ8cdOhpotFNVSeXJByIZ9iGNYTrWnIqUVcfiYyLwjMe58AJPiDvujmJtwT2omowRSX-r4-eMKjPPeuAR26zliEKDKJwYyAQktIgBN9yozmBn0PJ_n5ZdGfbMu_DLnHcvlWZ3i0pg	N.A

How to verify a JWT?

1. Get the OIDC Discovery Document
   a. Issuer URL + /.well-known/openid-configuration
      https://k8sou.apps.192-168-2-8.nip.io/auth/idp/k8sIdp/.well-known/openid-configuration
2. Get ***jwks_uri*** from the discovery document
   "jwks_uri":
   "https://k8sou.apps.192-168-2-8.nip.io/auth/idp/k8sIdp/certs",
3. Get the public key
   "n":
   "gDHMhjGEg5mSH-9AM68eGiebcBjgQ1WcQDaUk3rvqN9Gix_y-FNHL71wIYGpQuu4vg61ZF-IqVOmKqP1R001EhpNrc4kgu8huXoJmPexf2BN-IJjJSEtUwyscbLofppZORX-ben1mkXRCEWKCWsTzI6b0o75jWhBG13QTRYm4ZQANw7VJkGJc-sUmez0ETsI7qRDdK37pofvCQFfFpWboaswtN9017UCI2AqeoOu3PjkdITiEVTd5FZ0uiQDvJGRcj2TU-jI6ON9gbdMLGVvpnkAIqxLcmib4SYQB_IaP0A8E37K3o_YMcR_joIcfFDoSuoG09DOQAa17nTTVtIqCQ"

Risks of Bearer Tokens

- If leaked, impossible to revoke
- If permissions change, JWT is valid

Lab - Building, signing, & verifying an app container

1. Open Your Web Browser
2. https://k8sou.bsidesnova2022.tremolo.dev/
3. Follow along!

Authentication Anti-Patterns

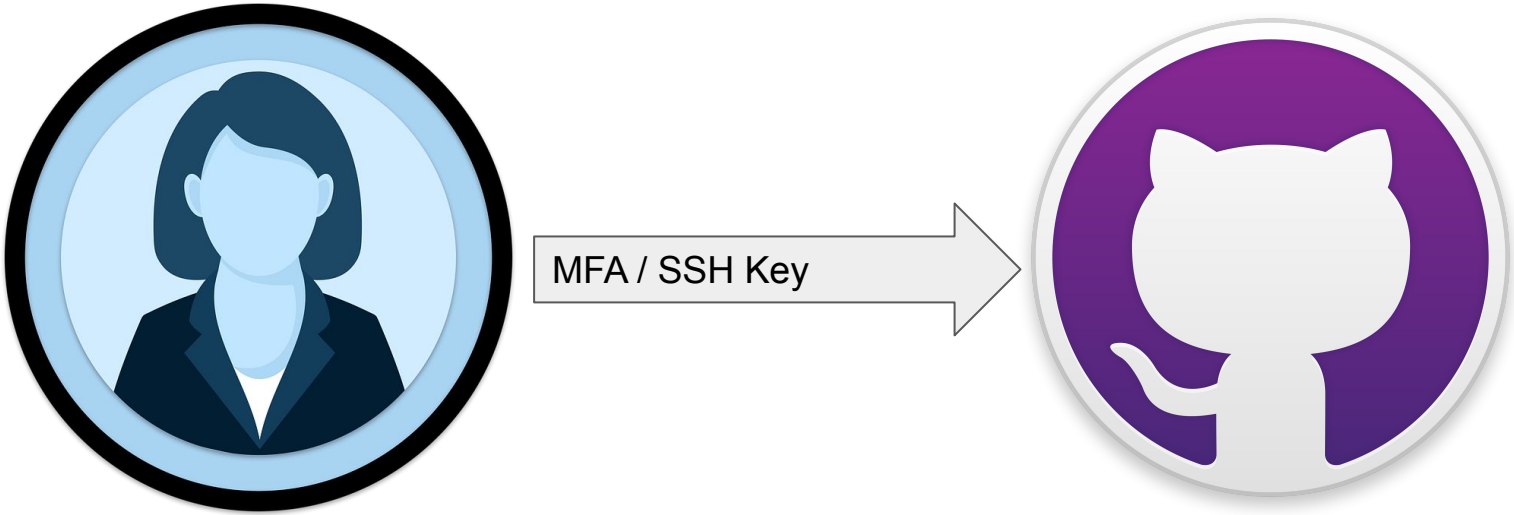
- Static tokens & passwords
- Personal access tokens
- Potentially X509 certificates

Signatures in Operations

- Live Lab - Verify container using admission controller
- Live Lab - Deploy unsigned container

What’s In an SBOM?

Identity across build infrastructure



Identity across build infrastructure

Unique JWT

Identity across build infrastructure

OpenSSH Keys

Identity across build infrastructure

Identity across build infrastructure

GitHub App

Identity across build infrastructure

Projected Token

Identity across build infrastructure

Static Token

Questions & Thank You!

@mlbiam / @CloudLvlMidnite