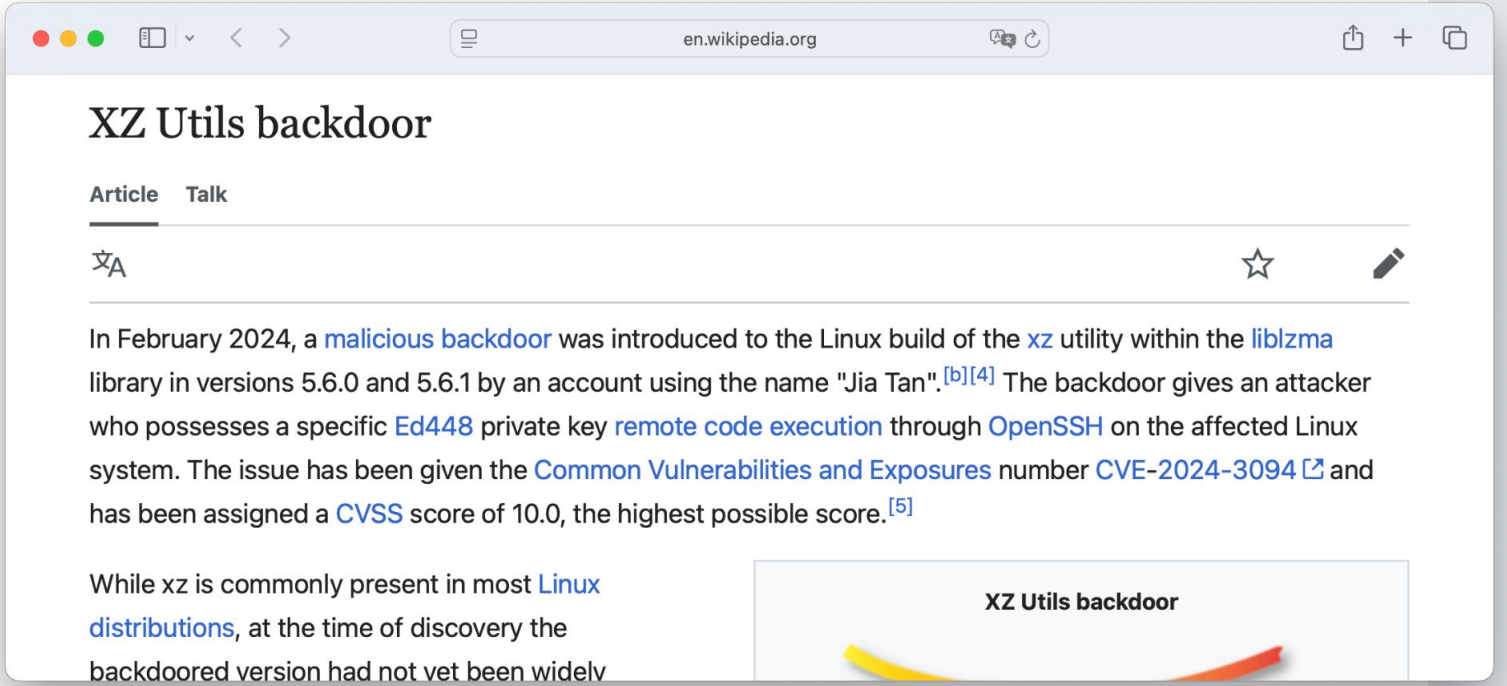
STREAMLINING SIGNED ARTIFACTS IN CONTAINER ECOSYSTEMS

Tõnis Tiigi

@tonistiigi
Docker

Why sign?


en.wikipedia.org
XZ Utils backdoor
Article
Talk
In February 2024, a malicious backdoor was introduced to the Linux build of the xz utility within the liblzma library in versions 5.6.0 and 5.6.1 by an account using the name "Jia Tan".[b][4] The backdoor gives an attacker who possesses a specific Ed448 private key remote code execution through OpenSSH on the affected Linux system. The issue has been given the Common Vulnerabilities and Exposures number CVE-2024-3094 and has been assigned a CVSS score of 10.0, the highest possible score.[5]
While xz is commonly present in most Linux distributions, at the time of discovery the backdoored version had not yet been widely
XZ Utils backdoor

Signing containers

Resist real practical attacks

- ➔ Users should clearly understand what the signature is protecting and what it is not protecting
- ➔ No signing just for some security checklist (new failure point)

Simple

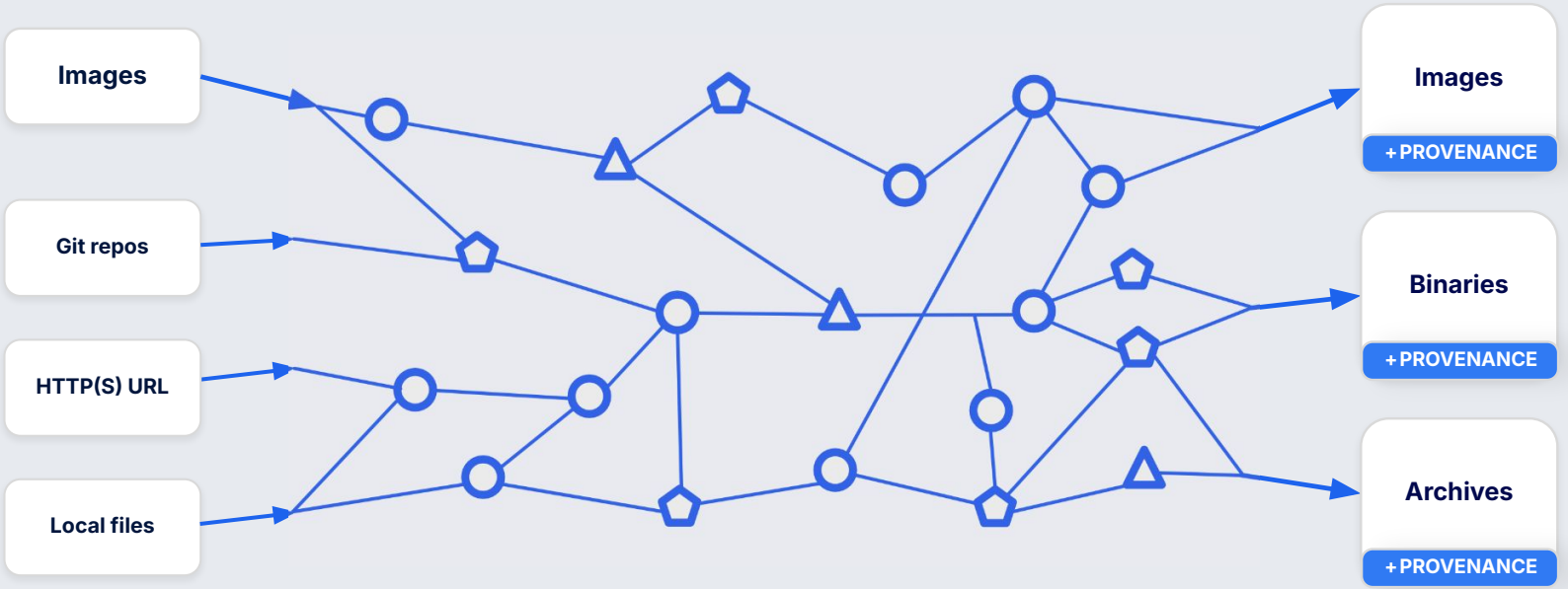
- ➔ Avoid key management, Yubikeys, key exchanges
- ➔ Only special security engineers can think about security protocols 24/7
- ➔ Integrations with existing tools
- ➔ High performance, friendly UX

BUILDKIT

docker

BuildKit

- ➔ Powers `docker build` , but not limited to Dockerfiles
- ➔ High performance
- ➔ Automatically caching
- ➔ Battle tested


Images
Git repos
HTTP(S) URL
Local files
Images
+PROVENANCE
Binaries
+PROVENANCE
Archives
+PROVENANCE

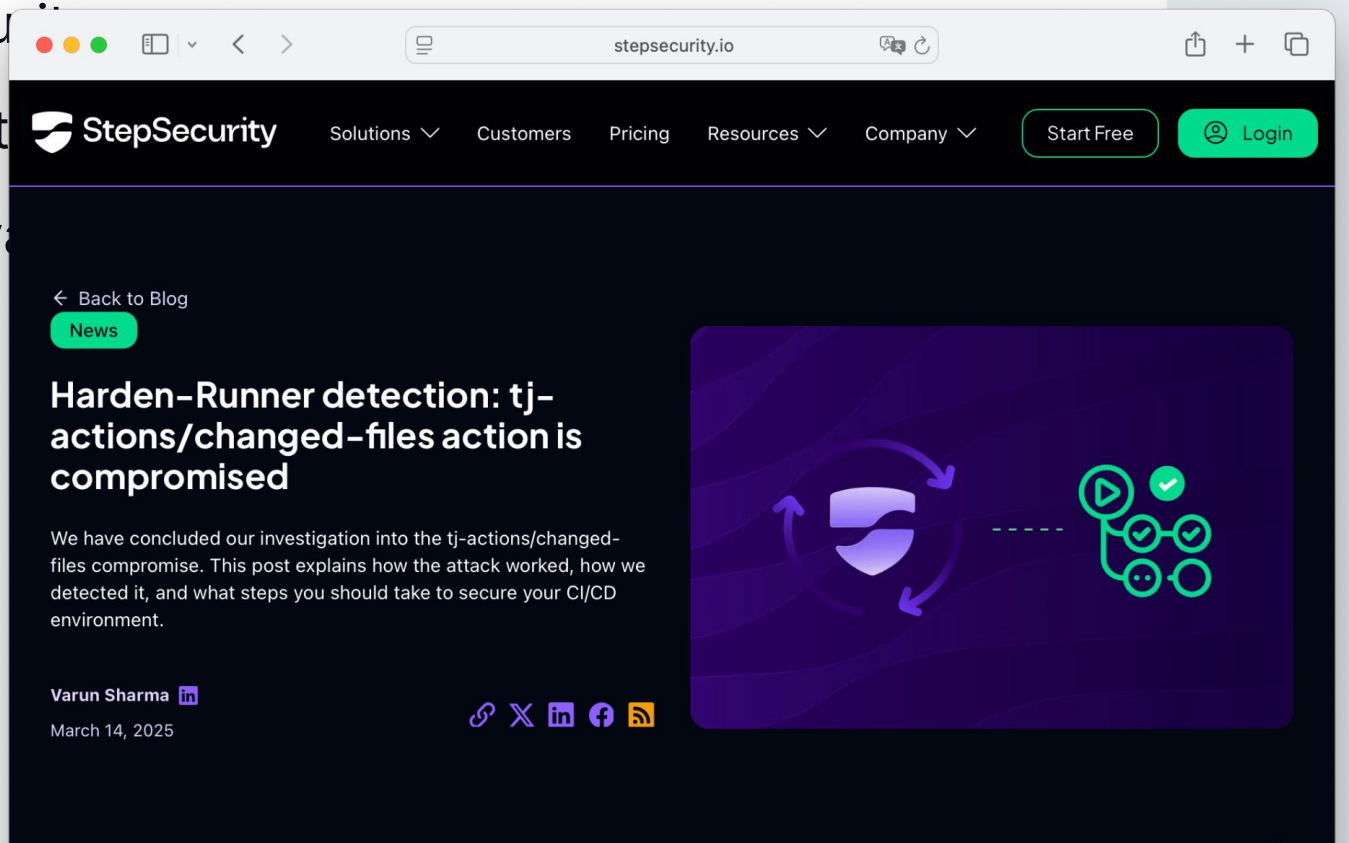
SLSA Provenance

- ➔ Snapshots of all build sources with checksums
- ➔ Build configuration
- ➔ Build steps definition
- ➔ Dockerfile source
- ➔ CI events
- ➔ Timing information
- ➔ Network access, reproducibility

SIGNING

Why not just checksums/lockfiles

- Good secu
- Lots of ext
- Hard to ev



Github docs example

```
      - name: Log in to the Container registry
        uses: docker/login-
action@65b78e6e13532edd9afa3aa52ac7964289d1a9c1
        with:
          registry: ${{ env.REGISTRY }}
          username: ${{ github.actor }}
          password: ${{ secrets.GITHUB_TOKEN }}


      - name: Extract metadata (tags, labels) for Docker
        id: meta
        uses: docker/metadata-
action@9ec57ed1fcdbf14dcef7dfbe97b2010124a938b7
        with:
          images: ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}


      - name: Build and push Docker image
        id: push
        uses: docker/build-push-
action@f2a1d5e99d037542a71f64918e516c093c6f3fc4
        with:
          context: .
          push: true
          tags: ${{ steps.meta.outputs.tags }}
          labels: ${{ steps.meta.outputs.labels }}
```

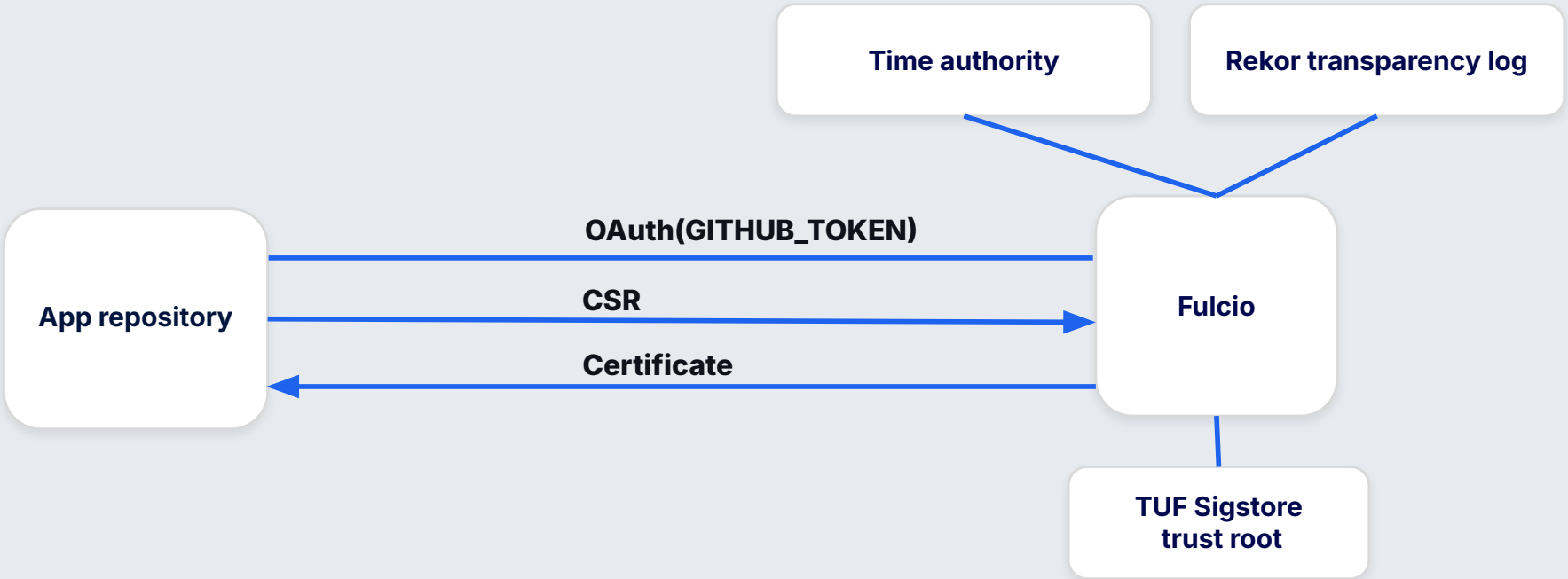
```
      - name: Generate artifact attestation
        uses: actions/attest-build-provenance@v3
        with:
          subject-name: ${{ env.REGISTRY }}/${{
env.IMAGE_NAME}}
          subject-digest: ${{ steps.push.outputs.digest }}
          push-to-registry: true
```

This step generates an artifact attestation for the image, which is an unforgeable statement about where and how it was built. It increases supply chain security for people who consume the image. For more information, see Using artifact attestations to establish provenance for builds.

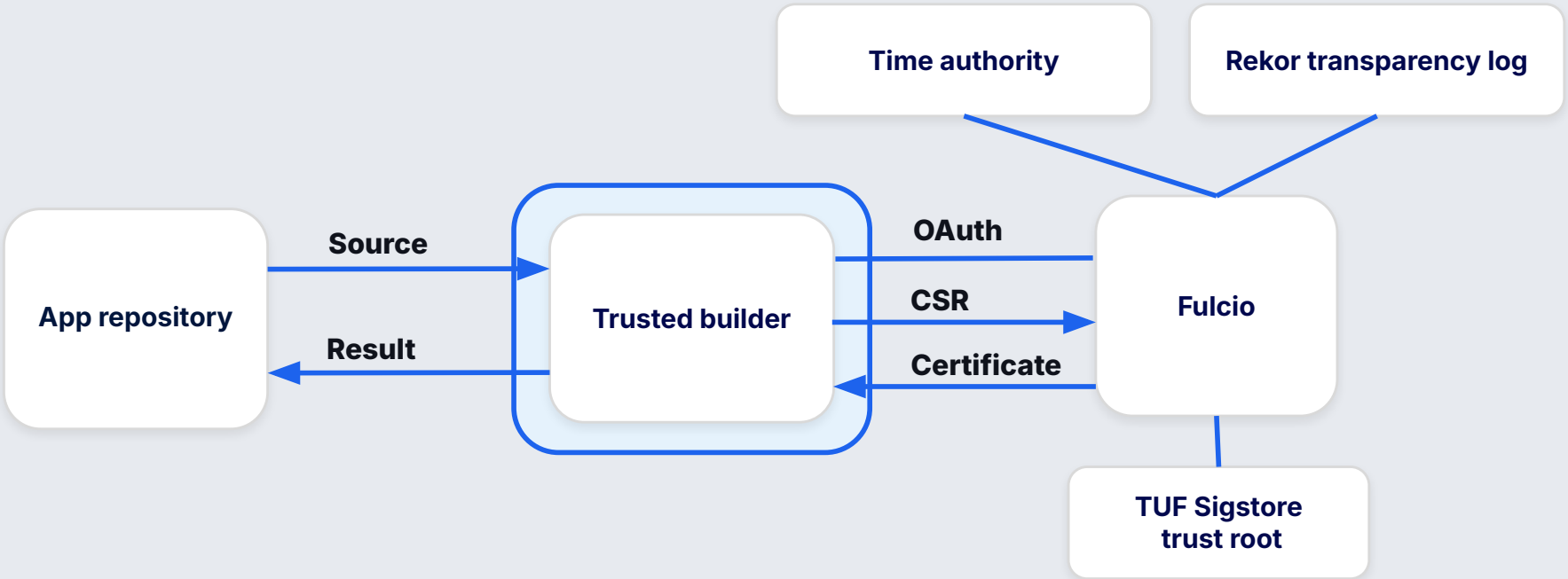
github.com/docker/github-builder

- ➔ Trusted reusable workflow from Docker
- ➔ Build results get Sigstore bundle signed attestations
- ➔ Only safe build configurations allowed
- ➔ Runs release versions of BuildKit, Buildx etc. (all verified)
- ➔ Optional signed cache
- ➔ Familiar UX from Docker Actions
- ➔ Other new features like distributed build

Sigstore simplified



Sigstore simplified



What does the signature prove?

Source

github.com/org/repo#ref

abcdef123

Artifact

sha256:6789fedcba

Protections: Registry credentials, Github credentials, GITHUB_TOKEN, Rogue maintainer, Build dependency compromise, Host environment leak

docker/github-builder

Trusted signing context.

Instance running verified BuildKit/Buildx releases.

moby/buildkit

Avoids any side-effects from environment or other build stages.

Captures detailed provenance about the build process.

VERIFYING

Verification often afterthought

https://docs.npmjs.com/generating-provenance-statements

Verifying provenance attestations

You can verify the provenance attestations of downloaded packages with the following `audit` command:

`npm audit signatures`

Example response showing the count of verified registry signatures and verified attestations for all of the packages in a project:

```
audited 1267 packages in 6s

1267 packages have verified registry signatures

74 packages have verified attestations
```

Because provenance attestations are such a new feature, security features may be added to (or changed in) the attestation format over time. To ensure that you're always able to verify attestation signatures check that you're running the latest version of the npm CLI. Please note this often means updating npm beyond the version that ships with Node.js.

Manual verification/inspection tools

- ➔ `docker inspect` now shows verified signatures
- ➔ Cosign
- ➔ sigstore/policy-controller (K8s YAML)

What people want?

➔ Single knob --verified

Reality

➔ What signature provider?
➔ What version format?
➔ Immutable tags?
➔ Tag before or after release?
➔ Git tags signed? (by HSM?)
➔ Annotated git tags only?
➔ Network access allowed?
➔ Cache allowed?
➔ etc.

Rego (OPA) policies support in Buildx

- ➔ Matching policy for your Dockerfile, eg. **Dockerfile.rego** automatically loaded with build.
- ➔ All build sources need to pass policy for the build to continue.
  - ◆ Images (including signatures, attestations)
  - ◆ Git
  - ◆ URLs
- ➔ 50+ source properties to match against atm.

Example use cases

- ➔ Check attestation signatures
- ➔ Check image checksums
- ➔ Pin images to checksums
- ➔ Forbid specific versions/checksums
- ➔ Ensure immutable tags
- ➔ Disable local sources
- ➔ Allow only specific registries
- ➔ Tweak policy based on build args (e.g. production)
- ➔ Check Git signatures
- ➔ Check annotated tags
- ➔ Require provenance

etc. etc.

Github builder signature example

```
package docker

allow if {
  input.image.repo = "org/app"
  docker_github_builder_tag(input.image, "org/app", input.image.tag)
}
```

```
package docker

allow if {
    input.image.repo = "org/app"
    some sig in image.signatures
    sig.type == "bundle-v0.3"
    sig.signer.certificateIssuer == "CN=sigstore-intermediate,O=sigstore.dev"
    sig.signer.issuer == "https://token.actions.githubusercontent.com"
    sig.signer.sourceRepositoryURI == "https://github.com/org/app"
    startswith(sig.signer.buildSignerURI, "https://github.com/docker/github-builder/.github/workflows/bake.yml@
    sig.signer.runnerEnvironment == "github-hosted"
    count(sig.timestamps) > 0
    sig.signer.sourceRepositoryRef == sprintf("refs/tags/%s", [input.image.tag])
}
```

Policy distribution

Preview Code Blame Raw

Verifying release integrity

Latest `xx` releases are built and signed using Docker Github Builder. You can verify the authenticity of the release with Rego policy in latest Docker Buildx.

```
is_xx if input.image.repo == "tonistiigi/xx"

is_xx_valid if {
  is_xx
  docker_github_builder_tag(input.image, input.image.repo, sprintf("v%s", [input
}
```

Optionally, you can also include shortlist of previous releases before signatures were introduced. `xx` version tags are always guaranteed to be immutable.

```
xx_releases = [
  {"version": "1.8.0", "checksum": "sha256:add602d55daca18914838a78221f6bbe42841
  {"version": "1.7.0", "checksum": "sha256:010d4b66aed389848b0694f91c7aaee9df59a
  {"version": "1.6.1", "checksum": "sha256:923441d7c25f1e2eb5789f82d987693c47b8e
  {"version": "1.5.0", "checksum": "sha256:0c6a569797744e45955f39d4f7538ac344bfb
]

is_xx_valid if {
```

RECAP

➔ No reason not to sign.

Hopefully soon no reason to trust unsigned images.

➔ Not all signatures are equal.

➔ Software pulling packages should verify pulled content.

- ➔ **docker/github-builder** available today
- ➔ Latest releases of BuildKit, Buildx all already built/signed with the same Github Builder.
- ➔ Experimental **Rego policy support** in latest Buildx

Thank you!