Securing the **JavaScript** Ecosystem with **Reproducibility**

tinyurl.com/**reproduce-2025**

Darcy Clarke

Based in **Toronto, Ontario, Canada** 🇨🇦

Founder & **CEO**
vlt /vōlt/

Member
OpenJS Foundation

Collaborator
Node.js WGs

npm · Themify · Deloitte. · TELUS · Google · Microsoft · CBS · Ducati · Porsche · Nike · Disney · Samsung · Panasonic · Burton · Red Bull · IAMS · Nestlé · The Lincoln Motor Company · facebook

vlt /vōlt/

vlt.sh

The **Team**

Darcy Clarke

CEO & Founder
Toronto 🇨🇦

npm · GitHub · Node.js · Themify

Isaac Schlueter

Principal Software Engineer
Oakland 🇺🇸

npm · GitHub · Node.js · yahoo!

Ruy Adorno

Staff Software Engineer
Montreal 🇨🇦

npm · GitHub · Node.js · Google

Luke Karrys

Staff Software Engineer
Phoenix 🇺🇸

GitHub · npm · Node.js · &yet

Jason Korol

Design Engineer
Toronto 🇨🇦

BrainStation

Jules Pichette

Business & Finance
San Francisco 🇺🇸

INOVIA · pwc · ATHLETA

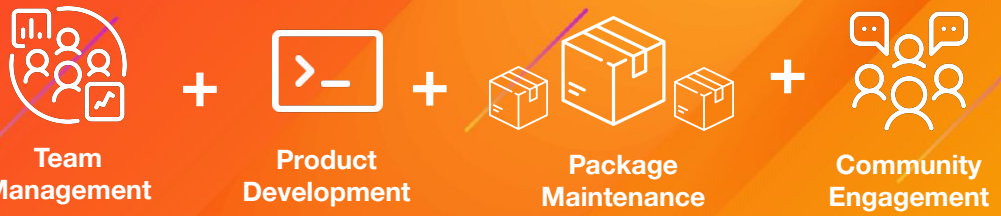
npm

3+ billion downloads /mo

~2% of **all** registry traffic

ex. **semver, tar, which, ini, ssri, write-file-atomic, hosted-git-info, make-fetch-happen** & more…

Staff Engineer Manager
July 2019 - December 2022 (~3.5yrs)



Securing the **JavaScript** Ecosystem with **Reproducibility**

tinyurl.com/**reproduce-2025**

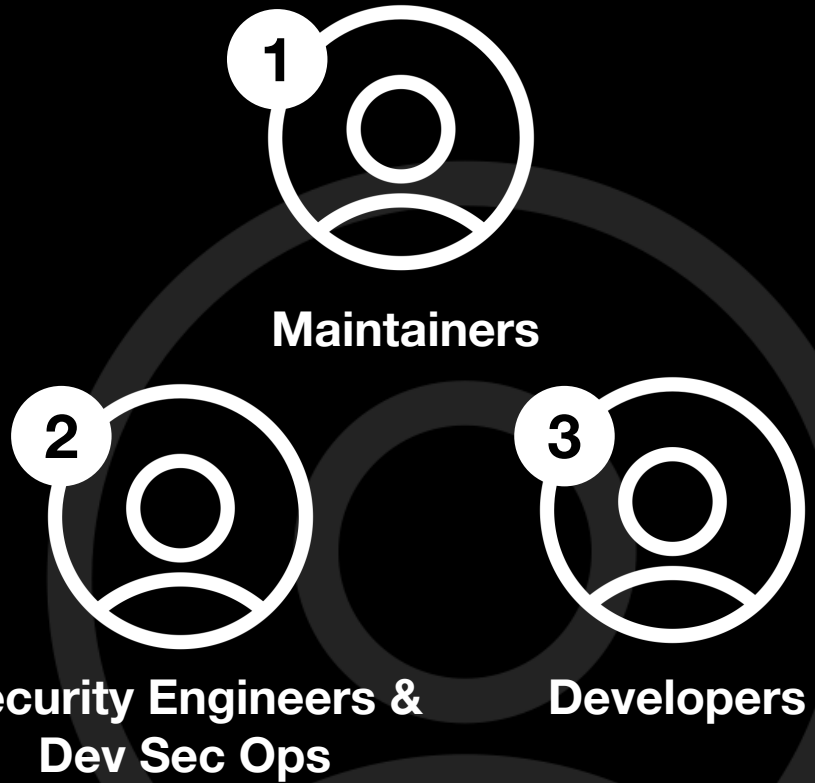
Please give feedback!

Overview

- **Trust**
  - **State of** Our Ecosystem
- **Provenance**
  - **Origins** of Tooling
- **Reproducibility**
  - **New** Tooling

Who

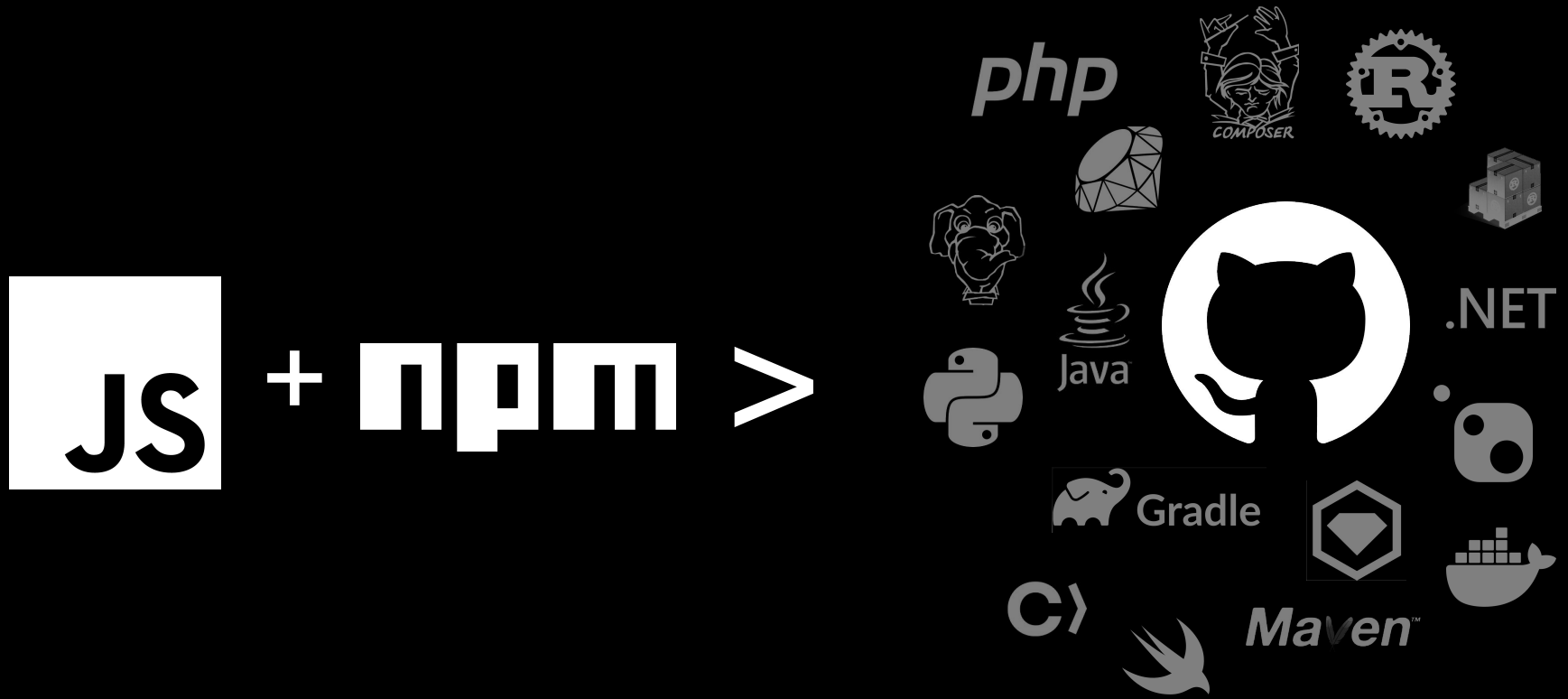
Are you?


1
Maintainers
2
Security Engineers &
Dev Sec Ops
3
Developers

Why, What & Who we **Trust?**

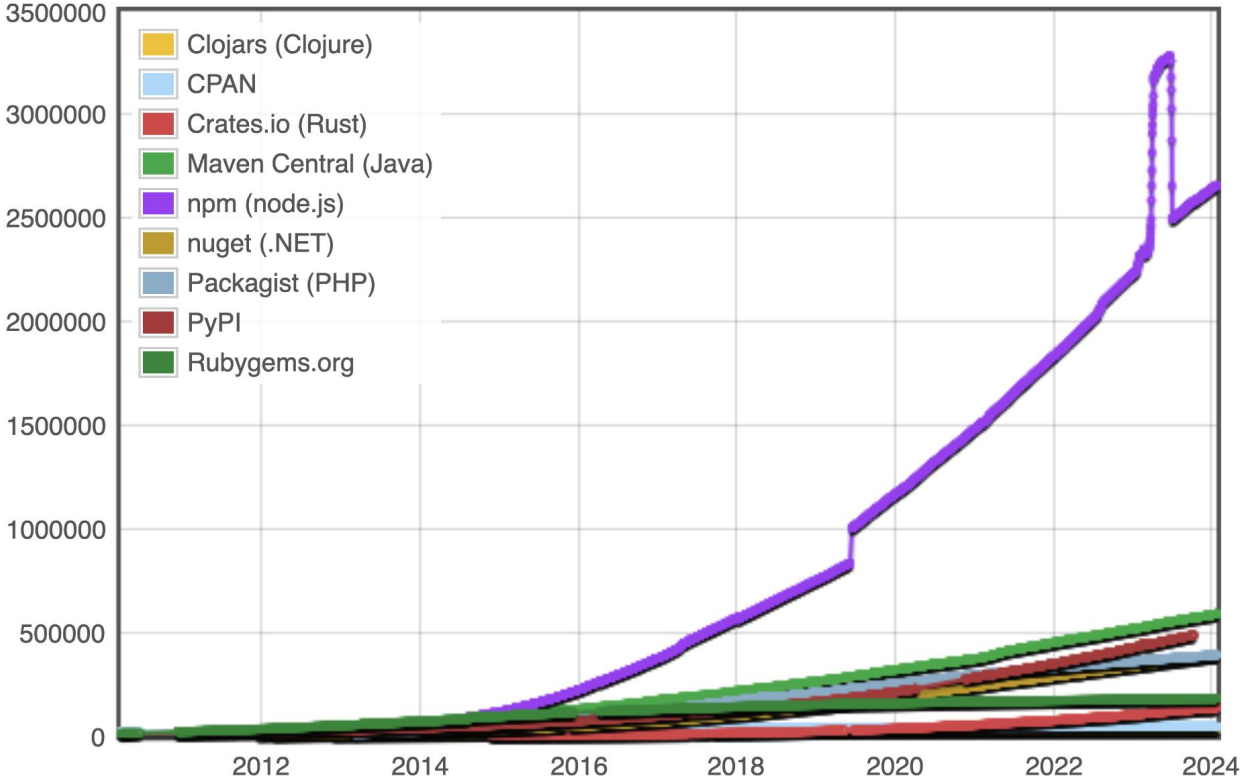
Who

Installs Dependencies?

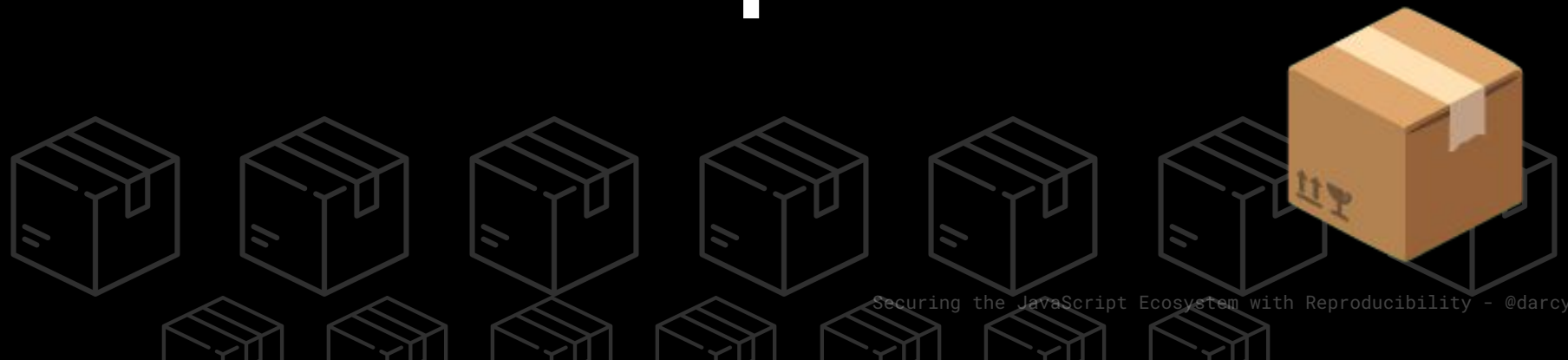

JS
+ npm >
php
COMPOSER
.NET
Java
Gradle
Maven

http://www.modulecounts.com/

Module Counts



JavaScript **projects** have
a lot of Dependencies

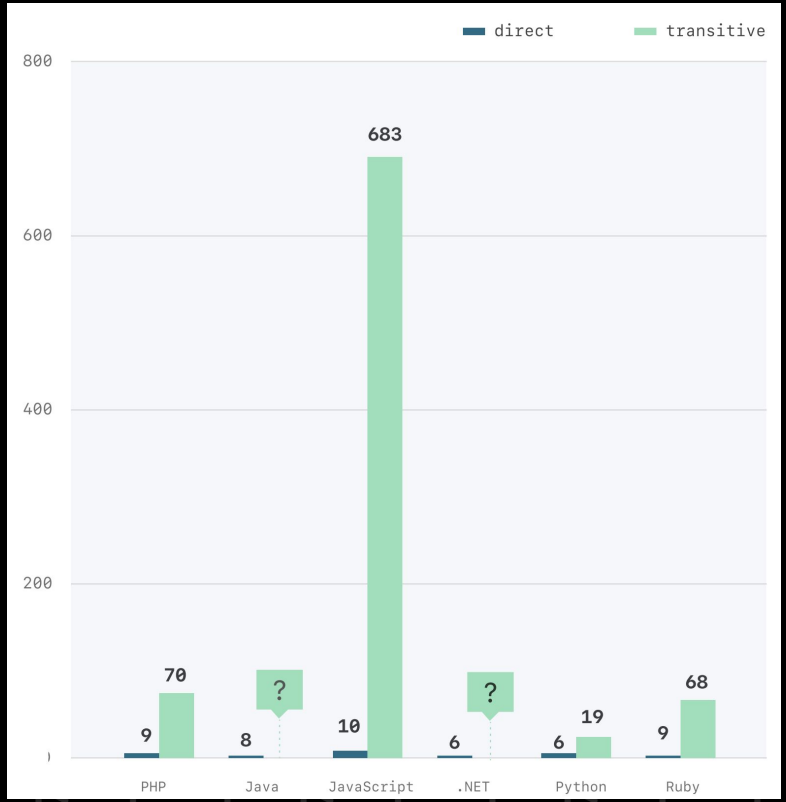


The average project has…

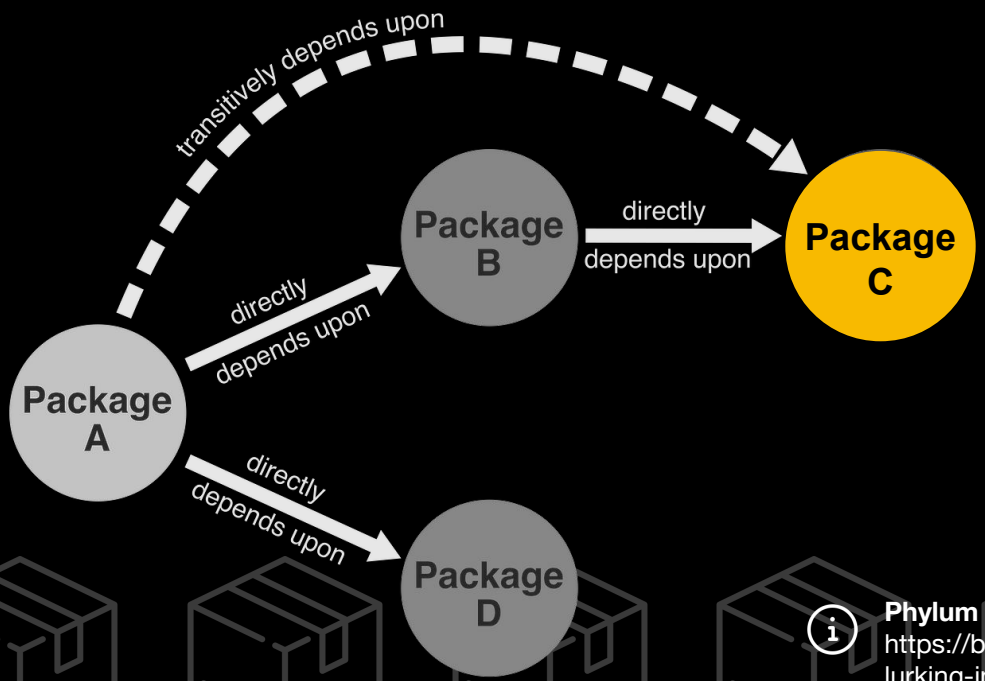
~683 transitive

dependencies

GitHub's State of the Octoverse:
https://octoverse.github.com/2021 & https://octoverse.github.com/2020

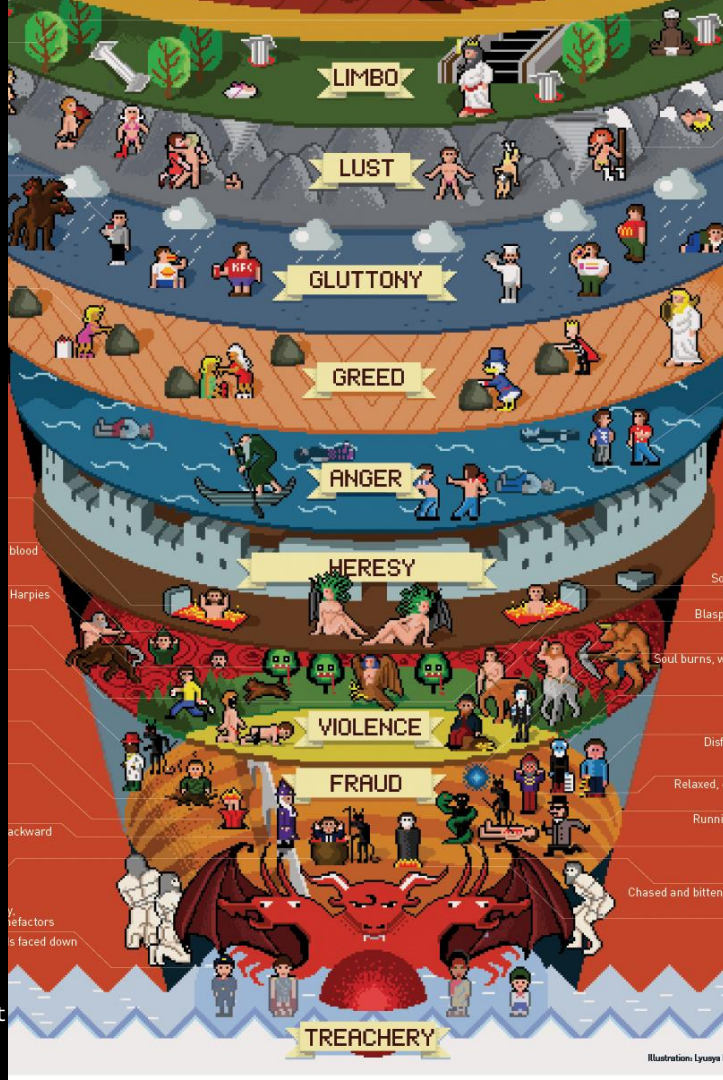


Transitive Dependencies



Phylum Blog Post
https://blog.phylum.io/hidden-dependencies-lurking-in-the-software-dependency-network

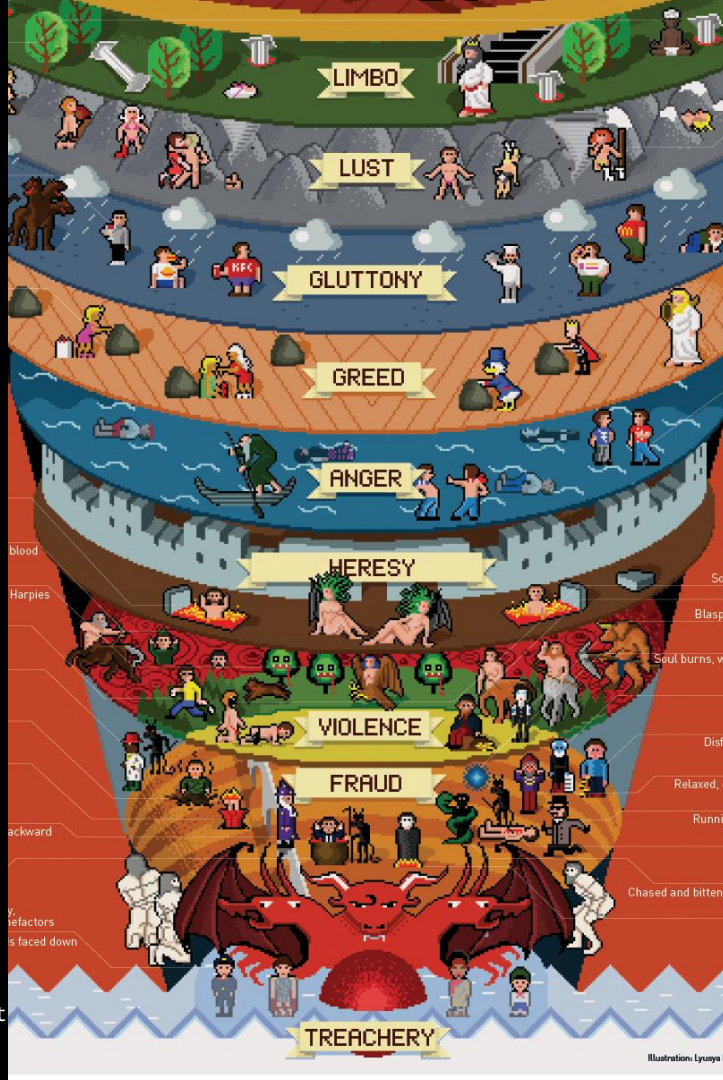
“Dependency Hell”



The 9 circles of

Dependency Hell

1. **Limbo** - **No** dependencies
2. **Lust** - **Any** dependency
3. **Gluttony** - **Every** dependency
4. **Greed** - **Never** contribute
5. **Anger** - **Protestware**
6. **Heresy** - Don’t follow **spec**
7. **Violence** - **Malware**
8. **Fraud** - **Grifts**
9. **Treachery** - **Rogues**



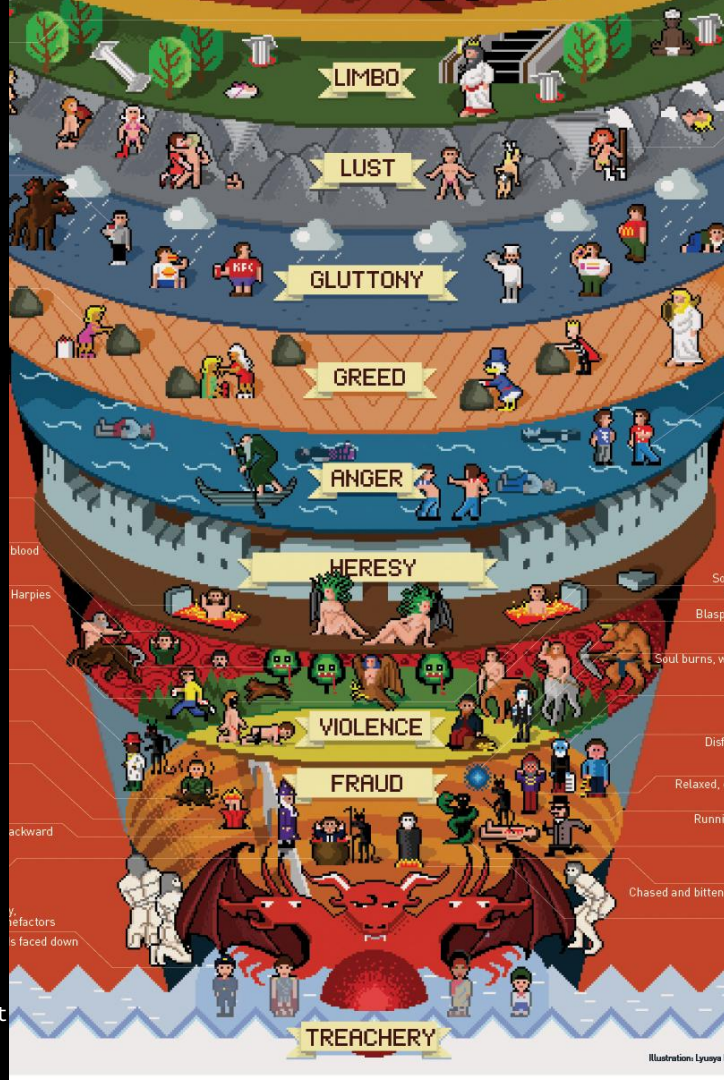
The 9 circles of

Dependency Hell

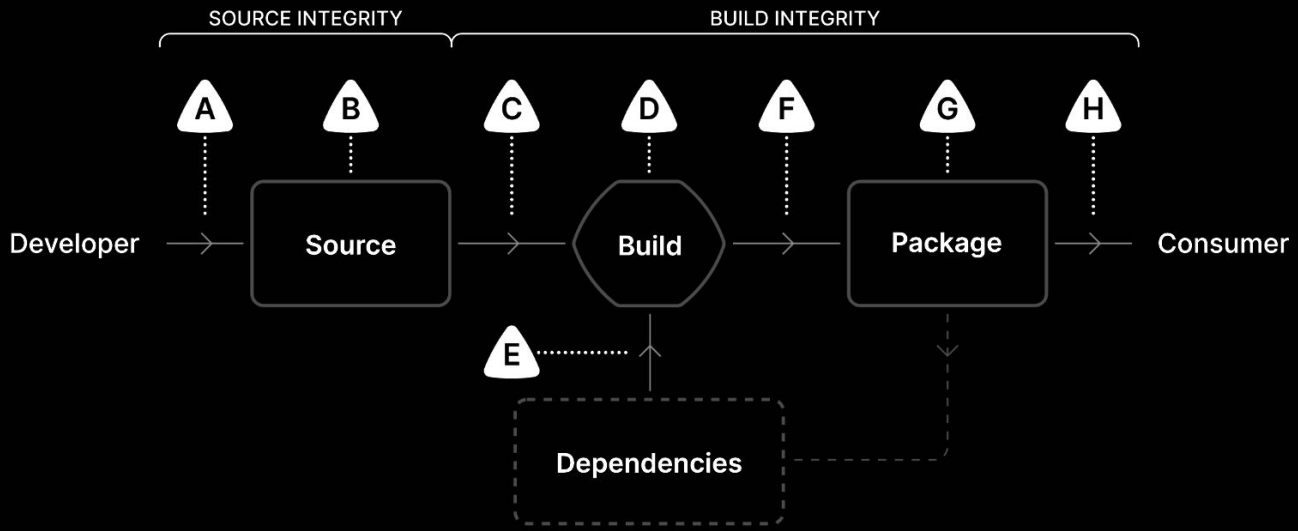
"*Circles of **Incontinence***"
ie. a "lack of self-control"

1. **Limbo** - **No** dependencies
2. **Lust** - Consuming **any** dependency
3. **Gluttony** - **Everything** is a dependency
4. **Greed** - **Consumption** w/ **little-to-no** contribution
5. **Anger** - Make **demands** of maintainers or consumers
6. **Heresy** - Don't follow **standards** / **specs** / **semver**
7. **Violence** - **Malware** / **Protestware**
8. **Fraud** - **Grifts** / **Social Engineering**
9. **Treachery** - **Rogue Actors** / **Trojans**

→ Supply-Chain Security **Nightmares** 🤦



Software Supply Chain Model



A Submit unauthorized change

B Compromise source repo

C Build from modified source

D Compromise build process

E Use compromised dependency

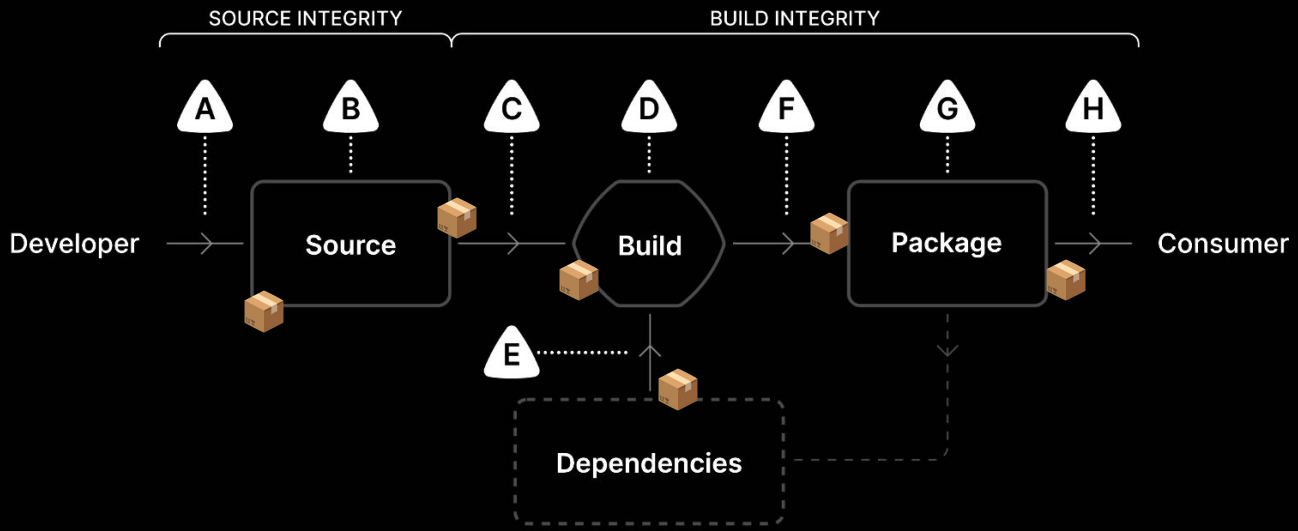
F Upload modified package

G Compromise package repo

H Use compromised package

Supply-chain Levels for Software Artifacts
https://slsa.dev/

Software Supply Chain Model



A Submit unauthorized change

B Compromise source repo

C Build from modified source

D Compromise build process

E Use compromised dependency

F Upload modified package

G Compromise package repo

H Use compromised package

Supply-chain Levels for Software Artifacts
https://slsa.dev/

JS

What is

Provenance?

What is

Provenance?

- **Cryptographic Link Between Source and Artifact**
  Provenance is a digitally signed proof that claims a package was built from a specific source repository and build process.
- **Flexible Metadata Attestations**
  Provenance allows any arbitrary metadata (e.g., build environment, dependencies, timestamps) to be signed and associated with an artifact, making it highly customizable.
- **Enhanced Software Supply Chain Observability**
  By tracing how software was built, provenance provides insights into who built it, where it was built, and how it was built.
- **Trust via Signing, Not Verification**
  Provenance mechanisms typically rely on trusted build environments (e.g., GitHub Actions, Buildkite) to sign attestations, but they don't inherently verify the artifact's contents.
- **Focus on Identity, Not Reproducibility**
  Provenance ensures that a specific entity (person, organization, or CI system) claims responsibility for a build, but it does not guarantee that the build can be independently reproduced.

Source? Artifact?

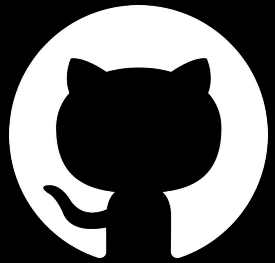
Why are there **Multiple Origins?**

https://**npmjs.com**

“packages”

tar + semver

VS.



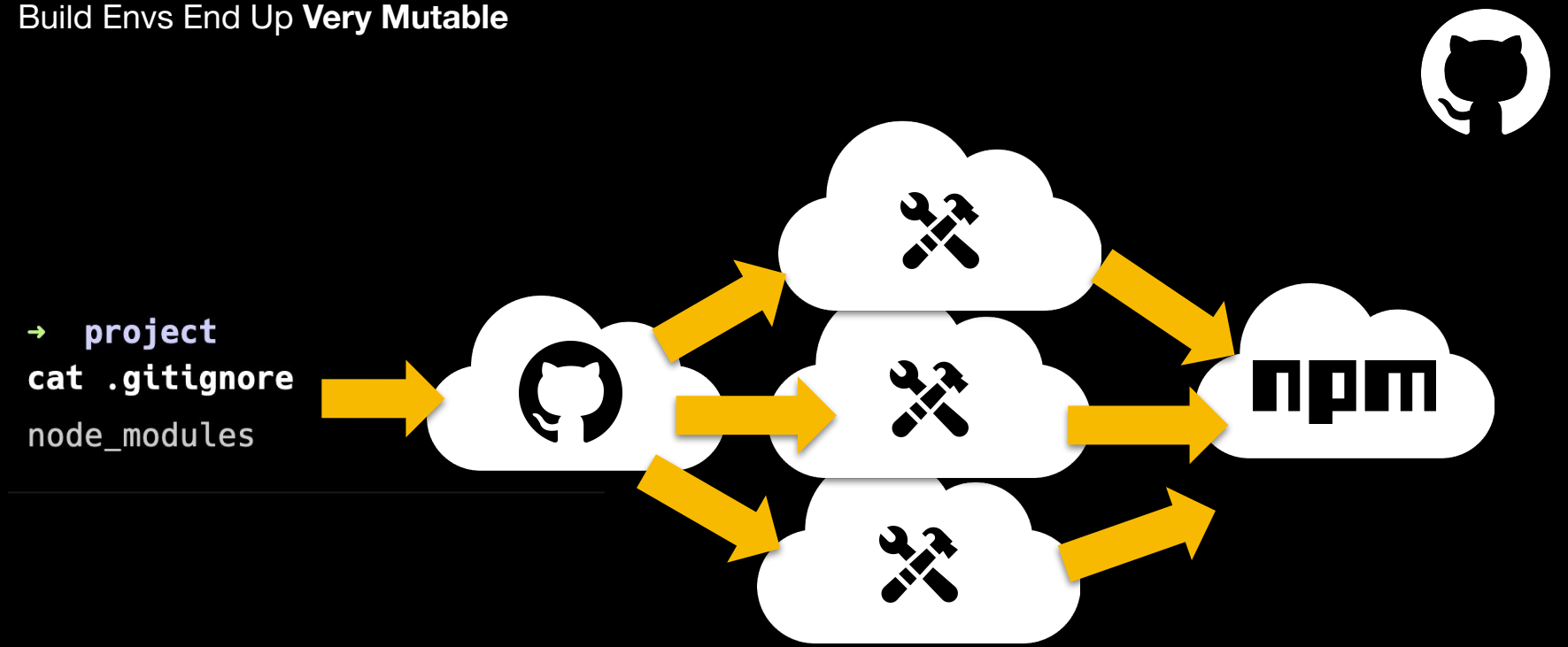
https://**github.com**

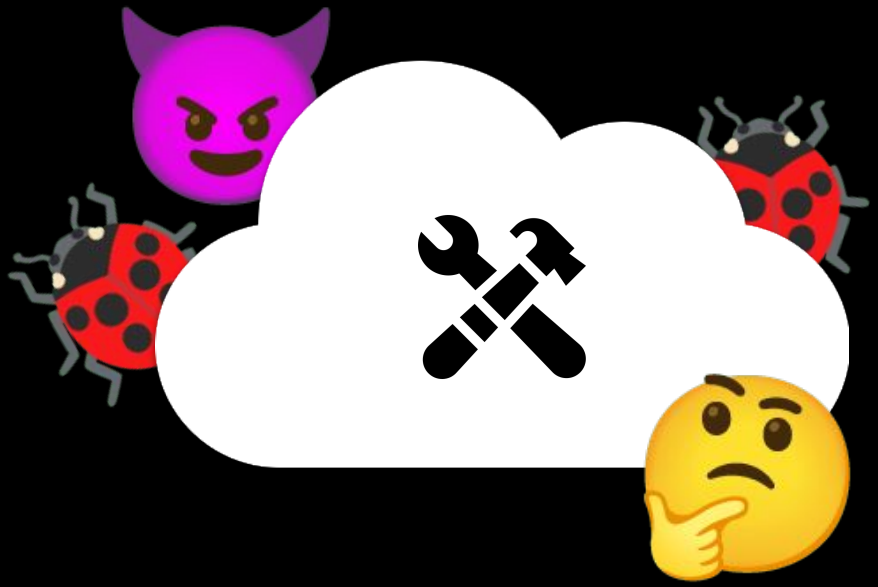
“repositories”

git

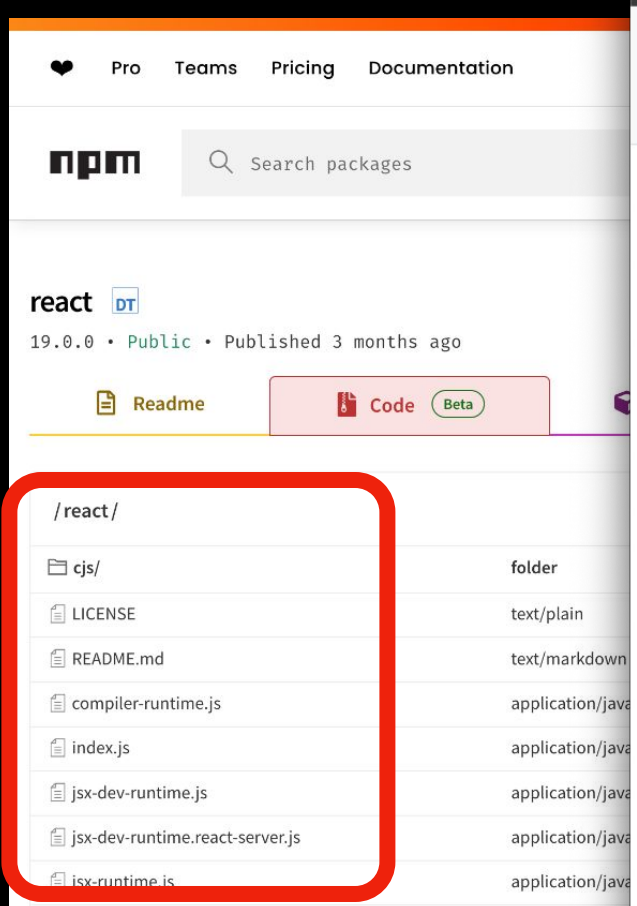
```
→  project
cat .gitignore
node_modules
```

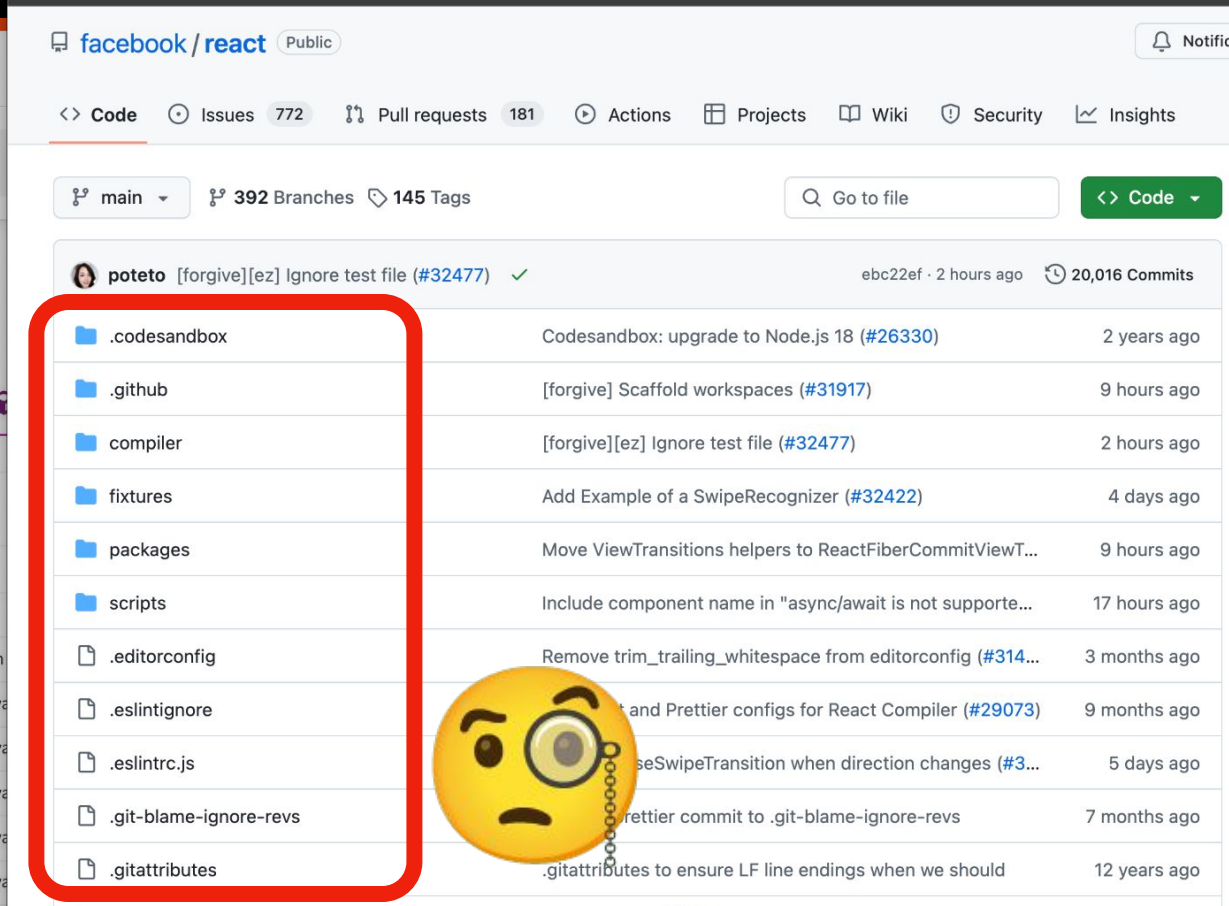
Build Envs End Up **Very Mutable**





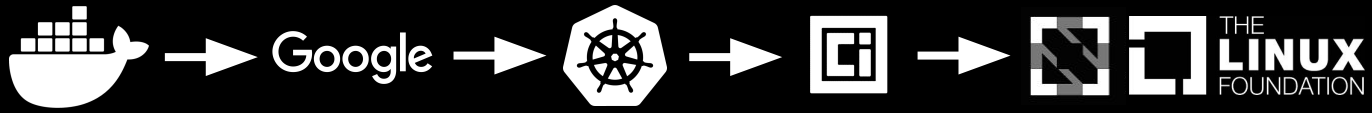
https://npmjs.com/package/**react** vs. https://github.com/**facebook/react**





What is **Sigstore?**

An Origin Story



solarwinds

Attack (March 2020)

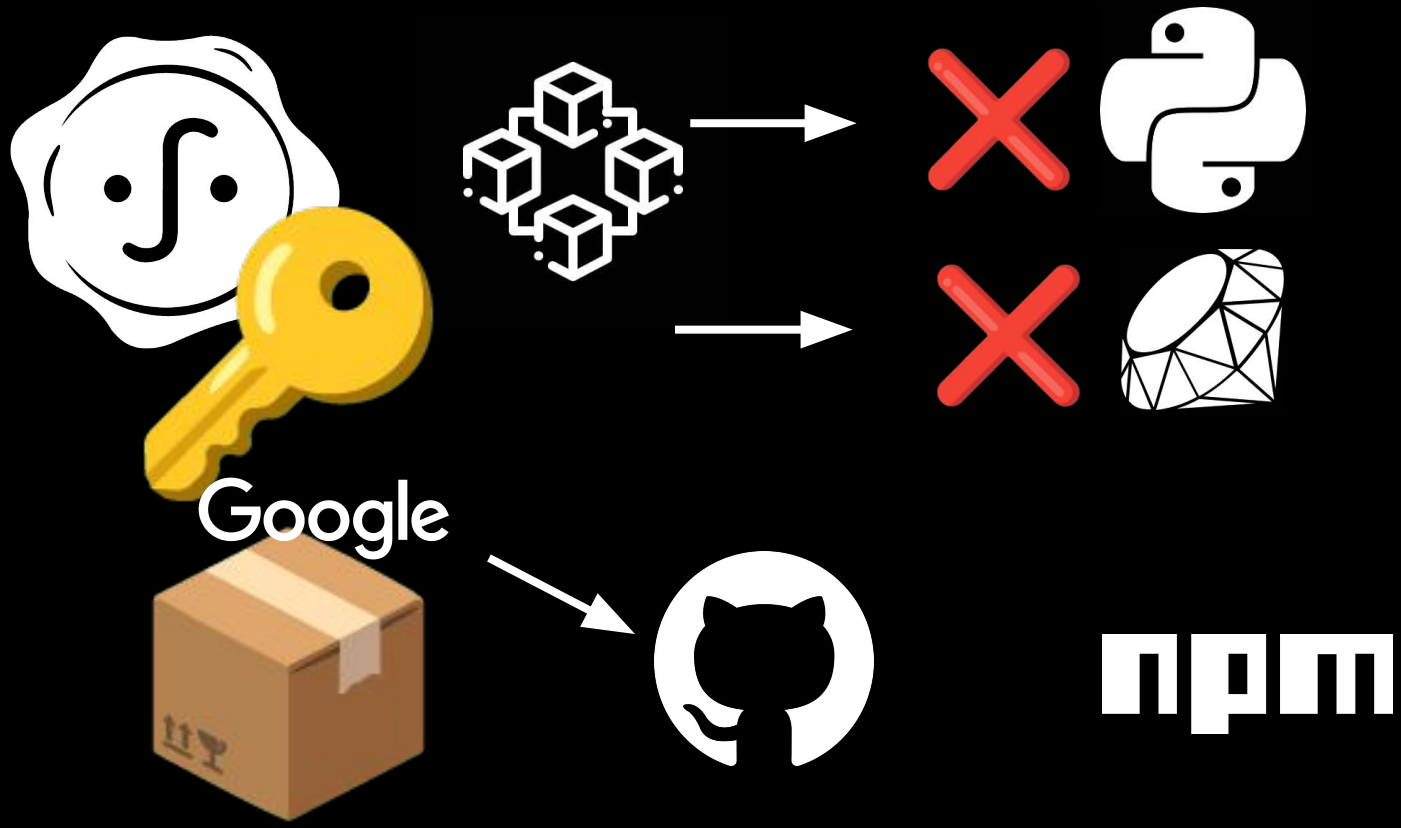
Microsoft vmware

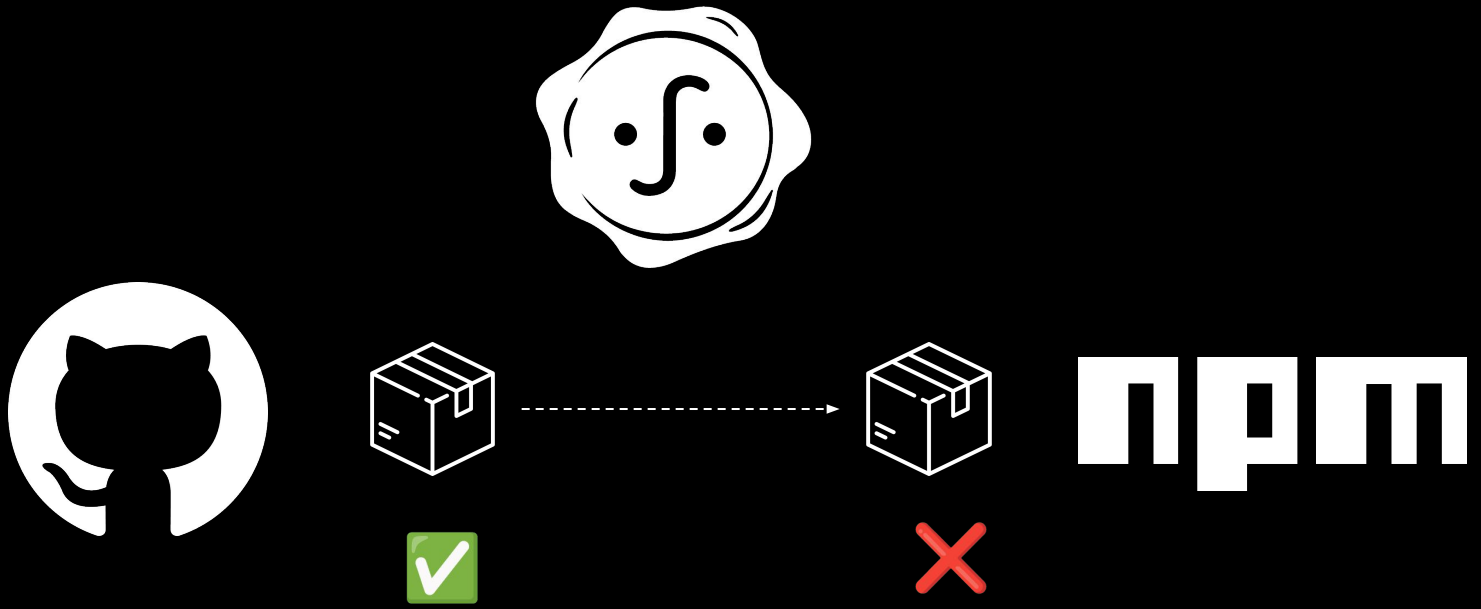
>200 Companies Affected (2 days later)

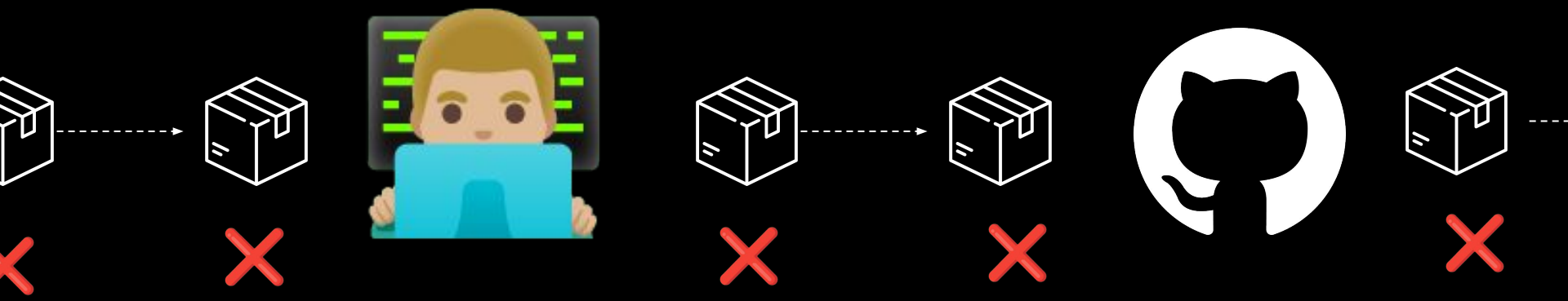
Biden Executive Order #14028 (May 2021)

Raises $10 Million (October 2021)

OpenSSF SLSA Chainguard **Sigstore**

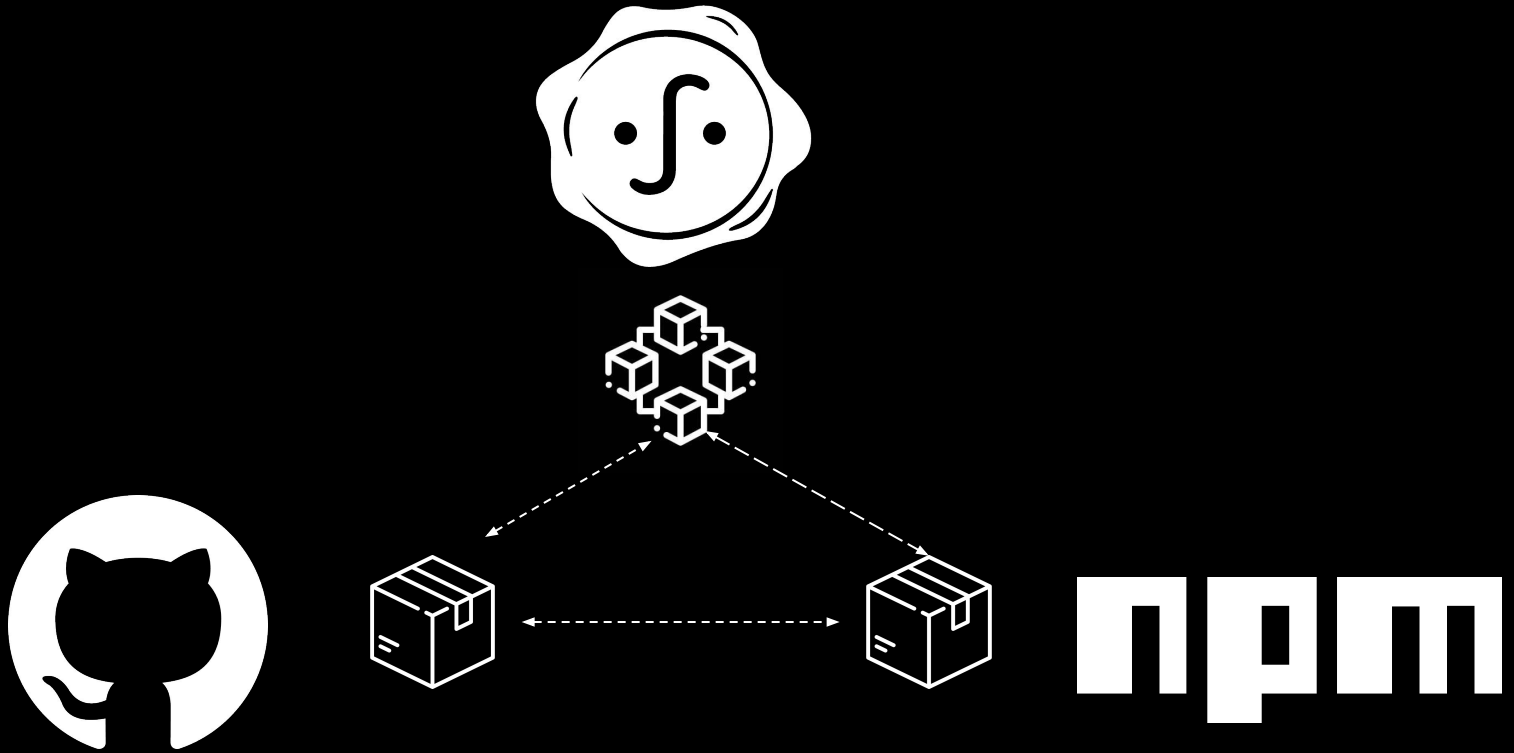

Google
npm


npm

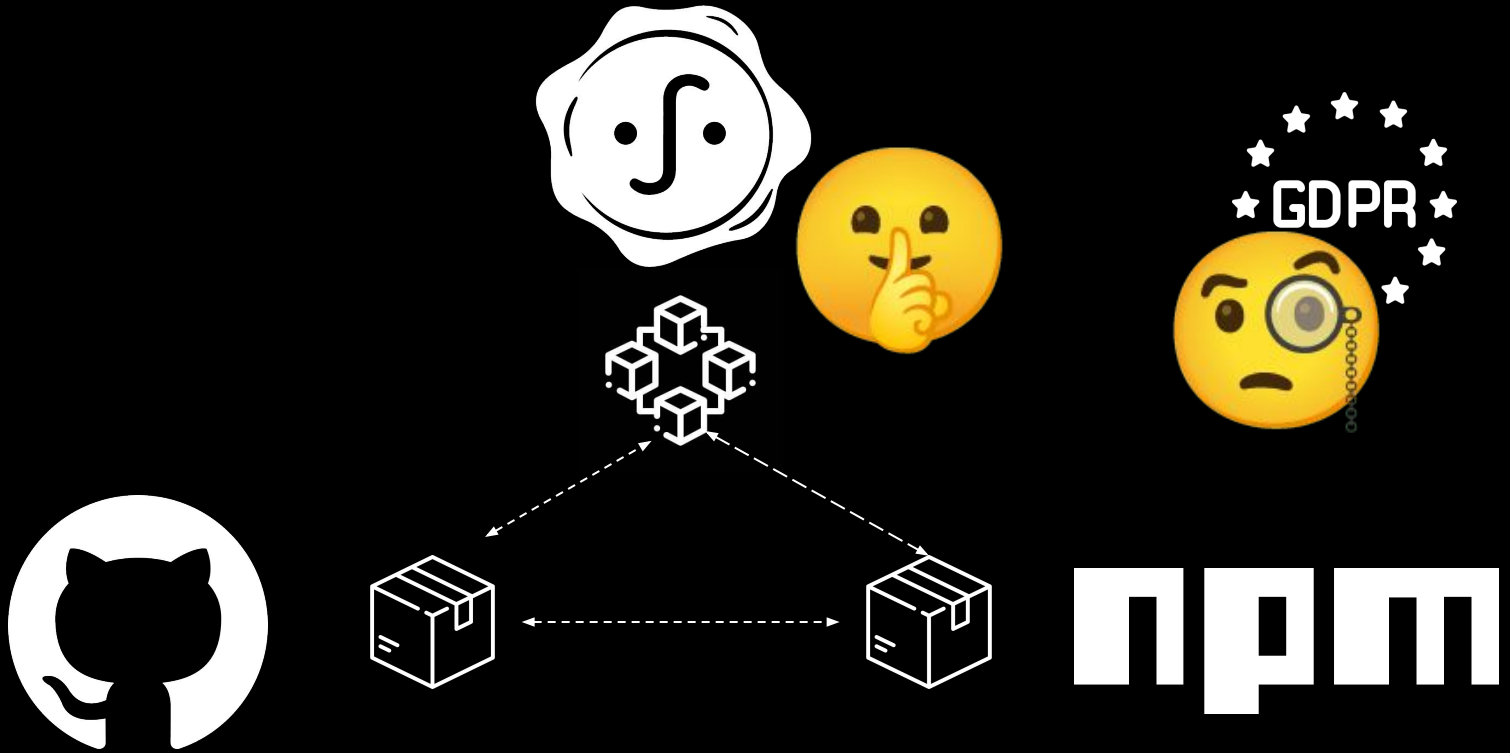


Where does it **end**?

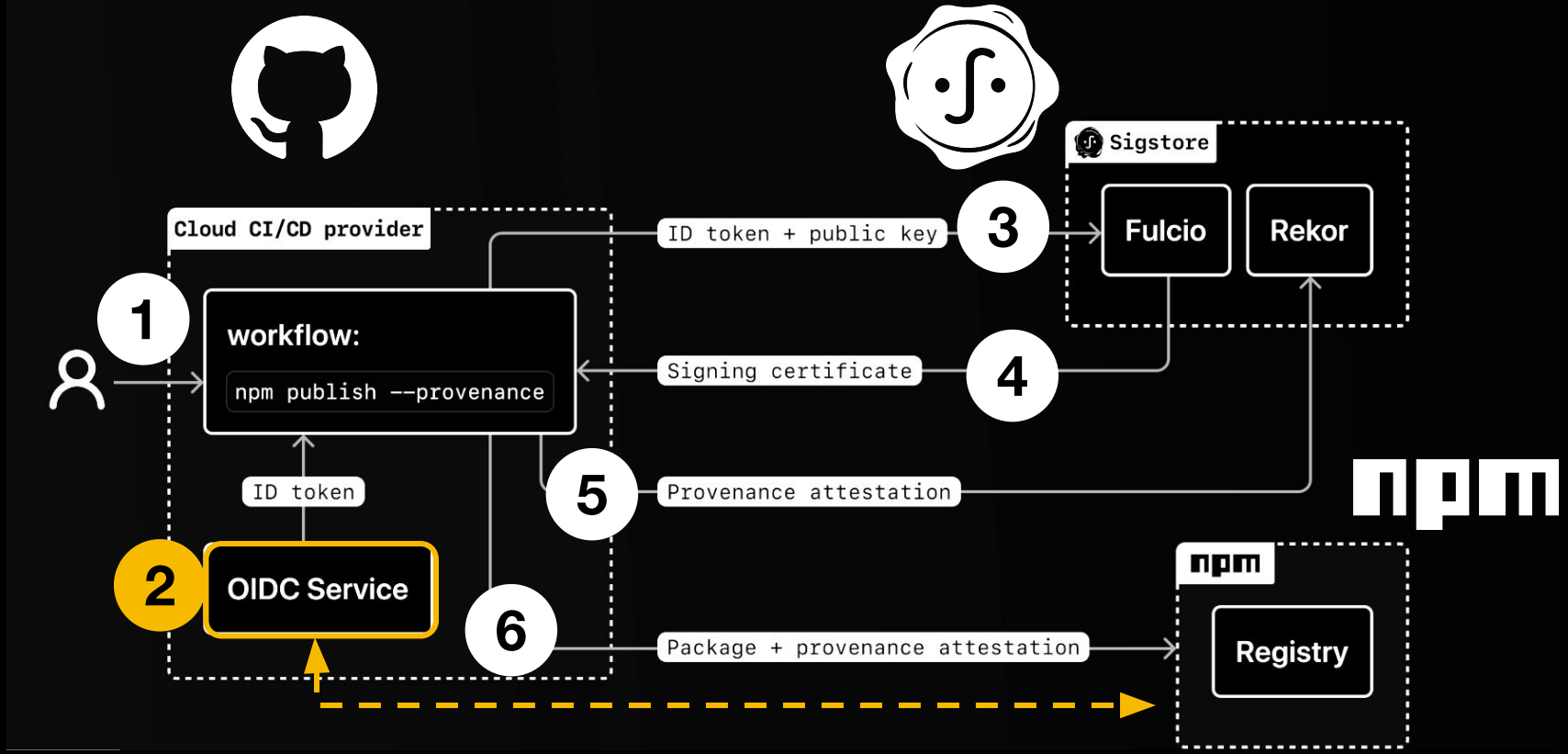
How it works…

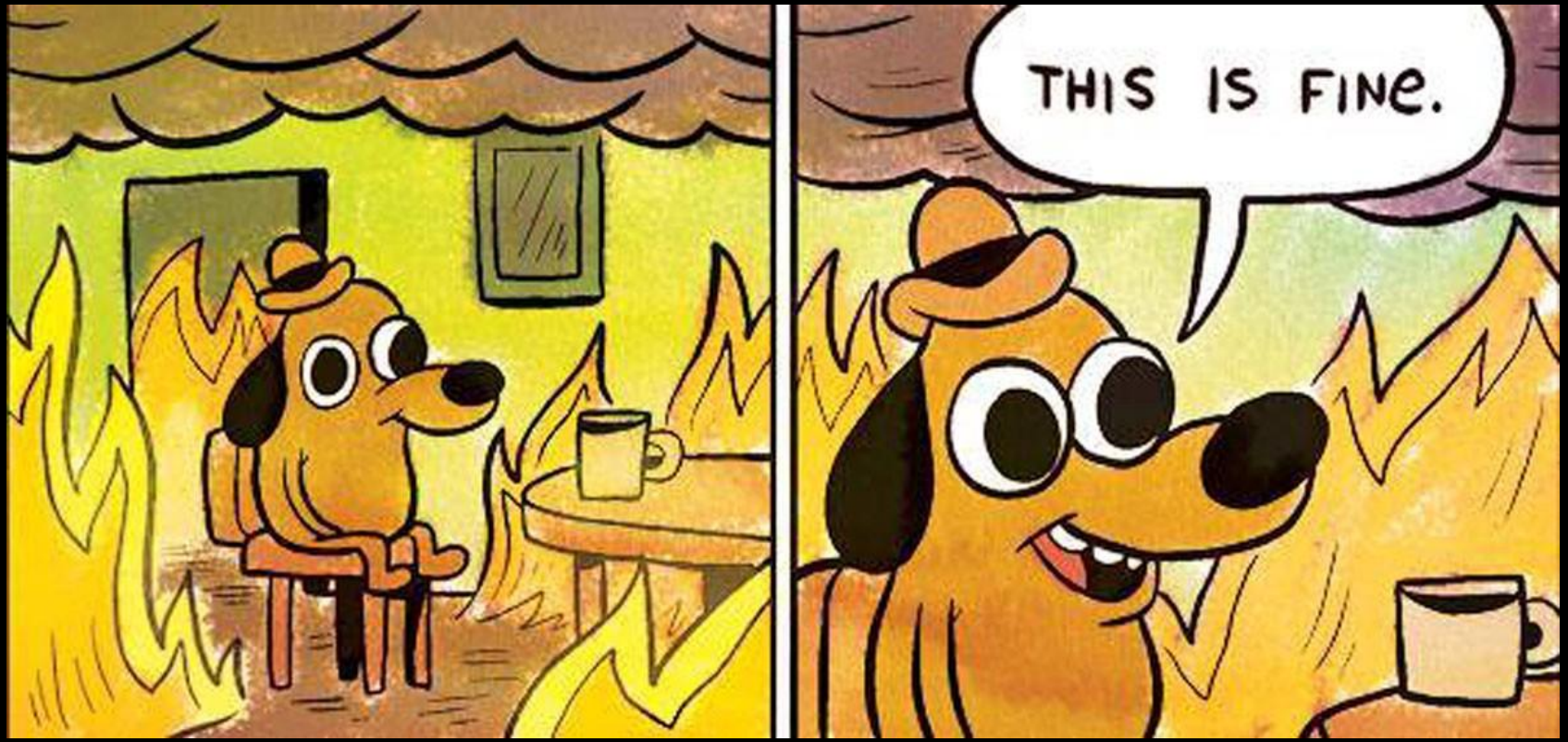


Compliant?

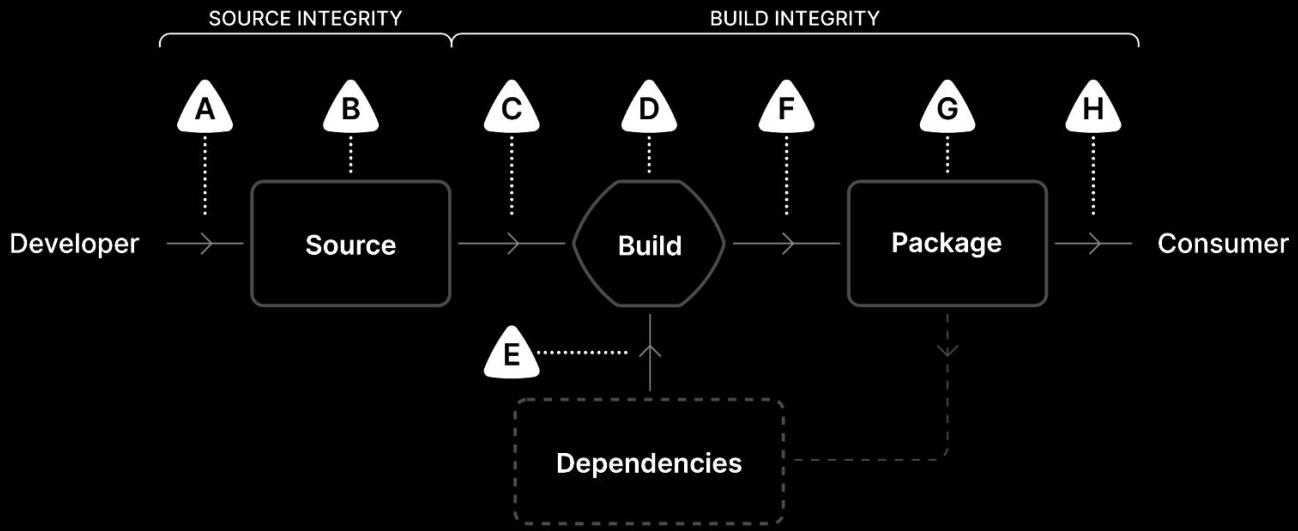


The Architecture




THIS IS FINE.

SLSA - Software Supply Chain Model



A Submit unauthorized change

B Compromise source repo

C Build from modified source

D Compromise build process

E Use compromised dependency

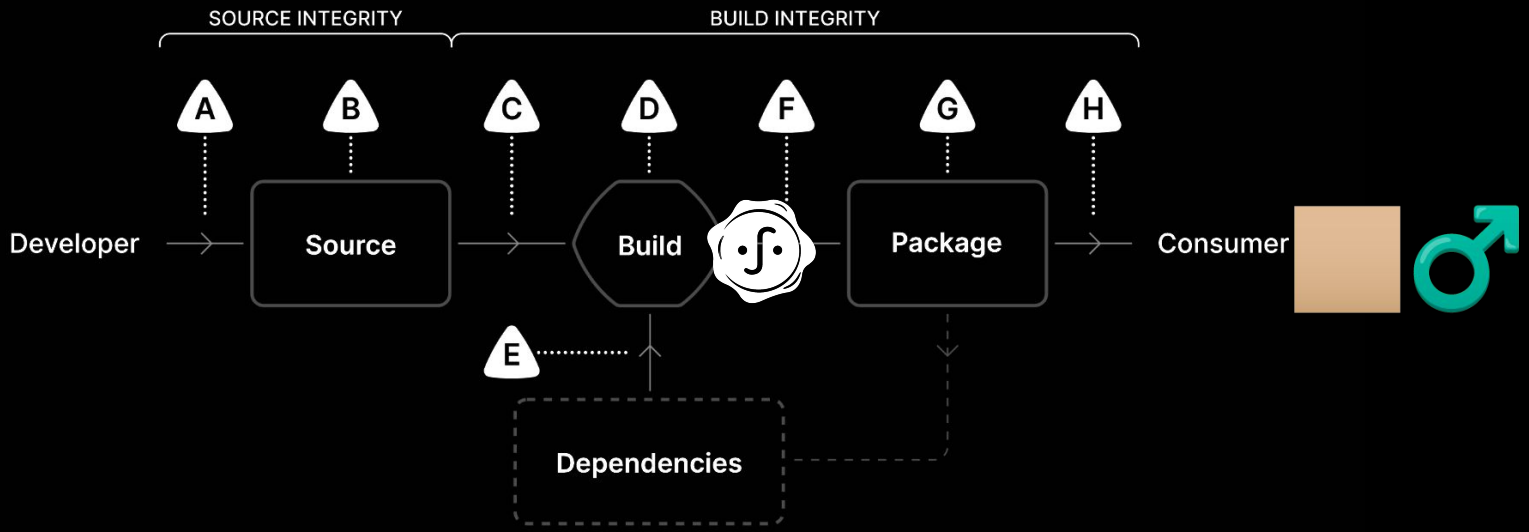
F Upload modified package

G Compromise package repo

H Use compromised package

Supply-chain Levels for Software Artifacts
https://slsa.dev/

SLSA - Software Supply Chain Model



A Submit unauthorized change

B Compromise source repo

C Build from modified source

D Compromise build process

E Use compromised dependency

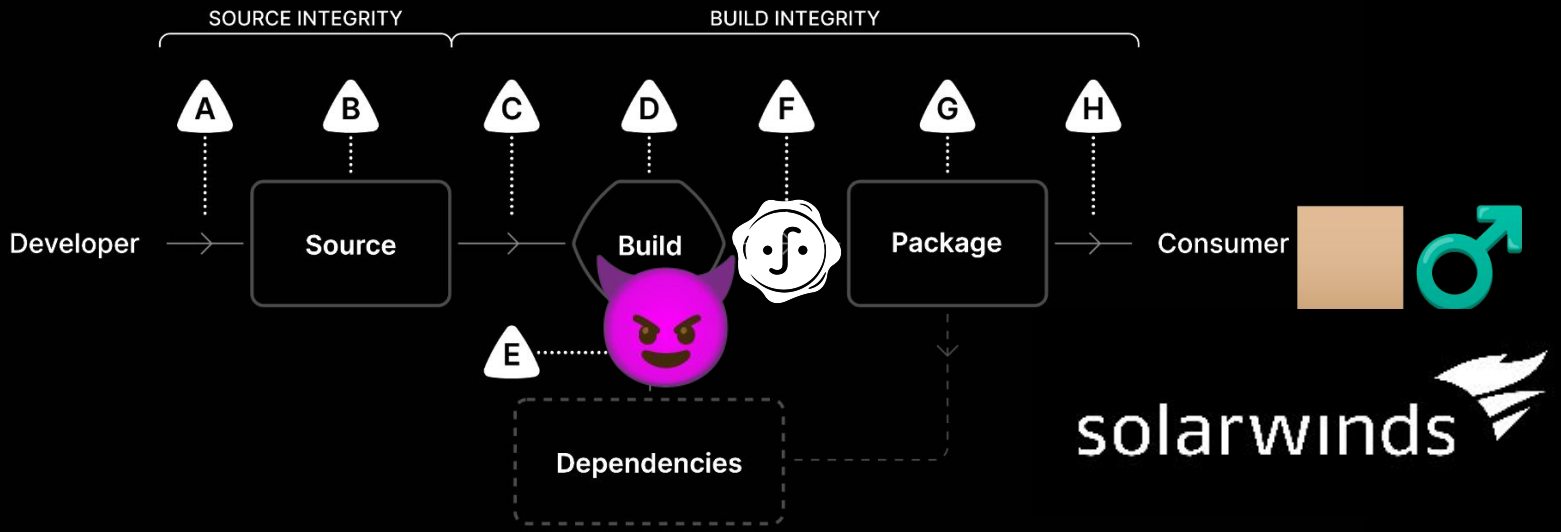
F Upload modified package

G Compromise package repo

H Use compromised package

Supply-chain Levels for Software Artifacts
https://slsa.dev/

SLSA - Software Supply Chain Model



A Submit unauthorized change

B Compromise source repo

C Build from modified source

D Compromise build process

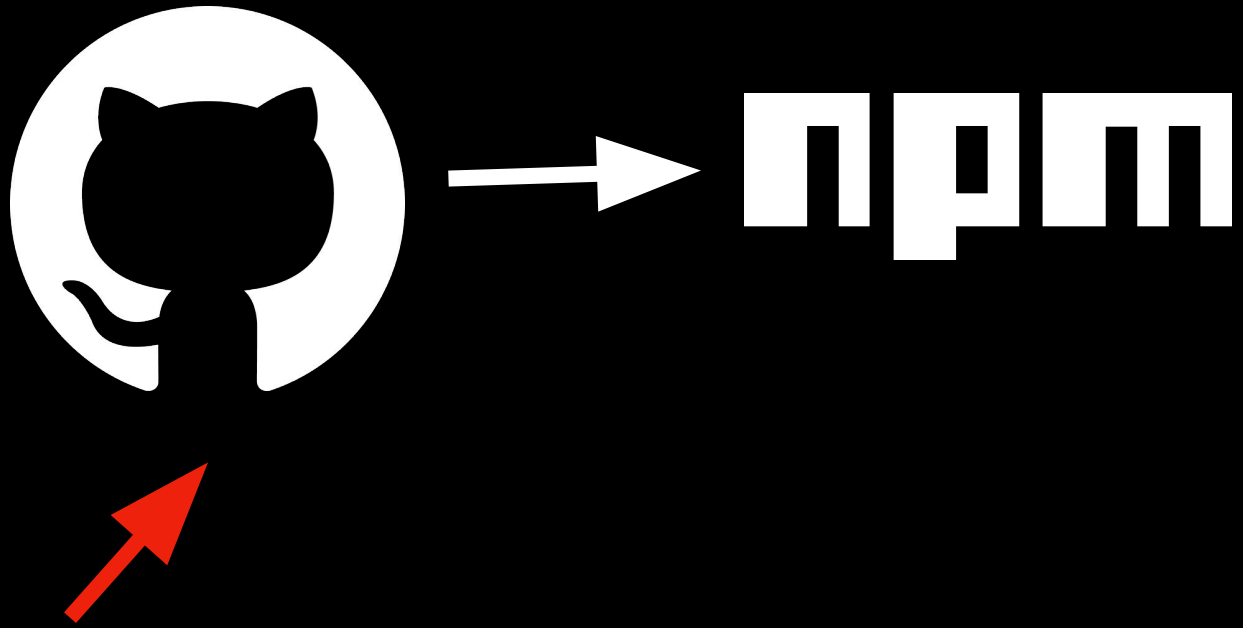
E Use compromised dependency

F Upload modified package

G Compromise package repo

H Use compromised package

Supply-chain Levels for Software Artifacts
https://slsa.dev/


npm

Other

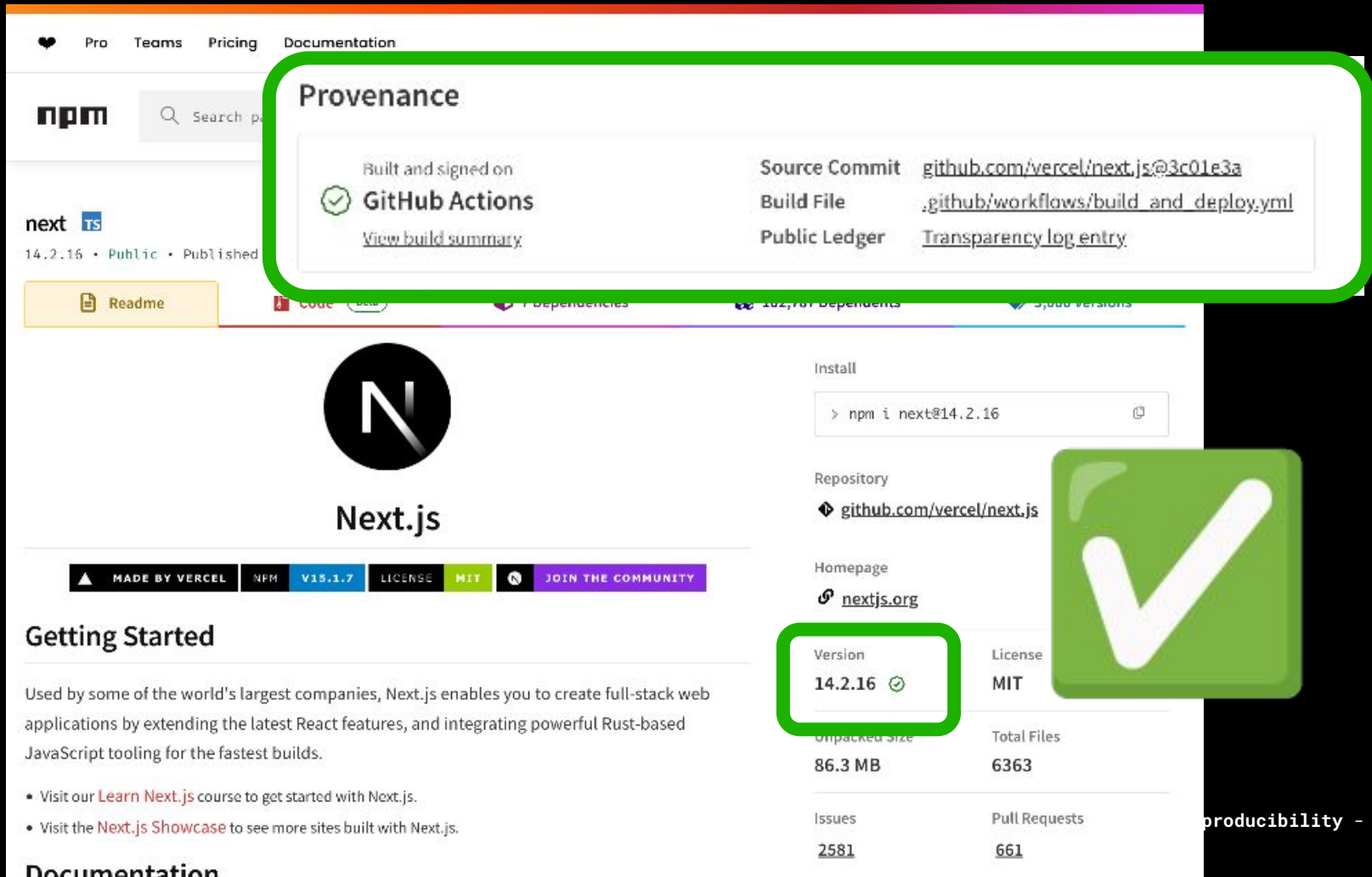
Provenance Issues

➡️ Forward-facing with **no way to backport**

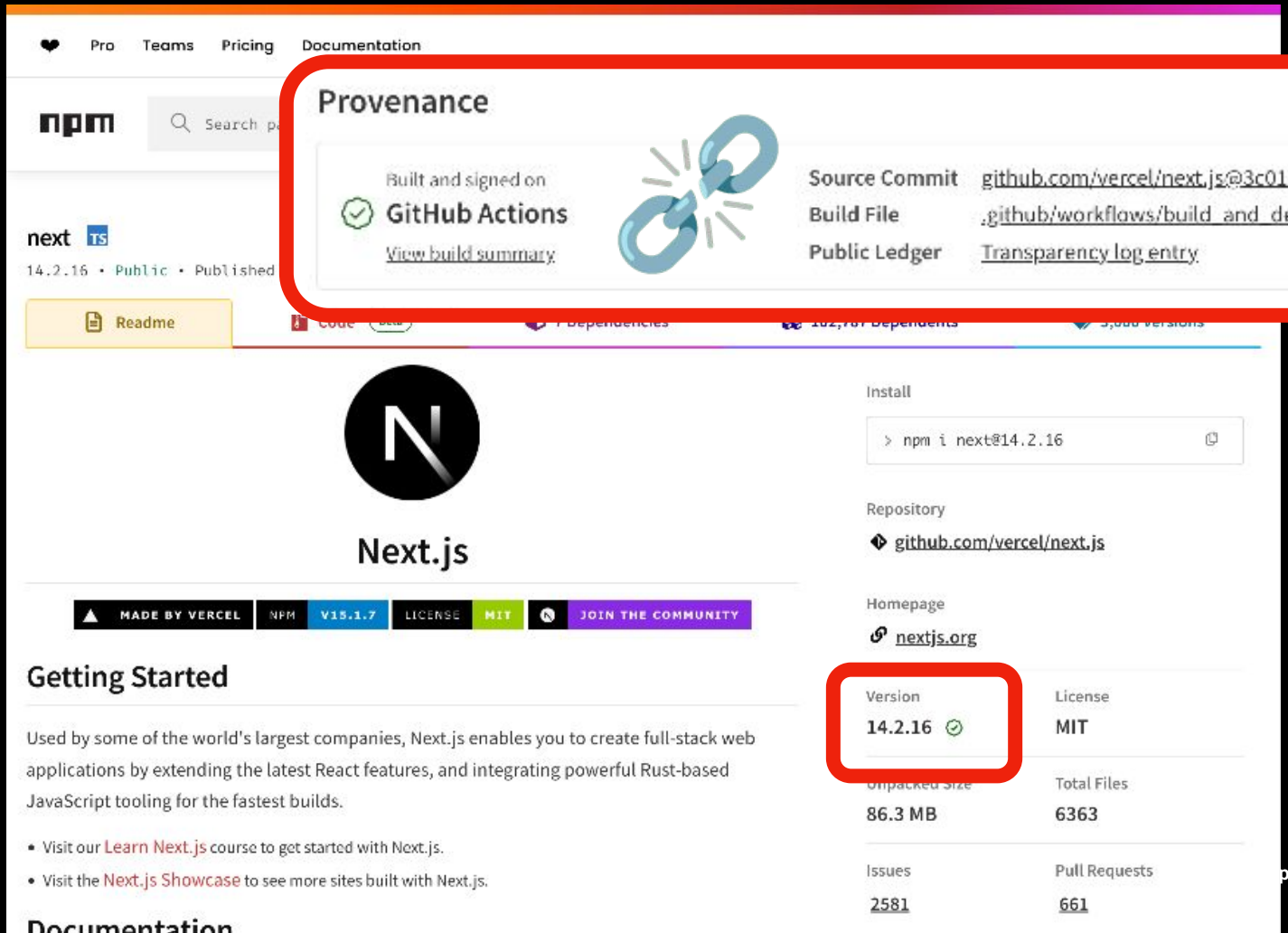
- M🟫💻iners **must change build processes**
  - ☁️ **Must** use cloud envs
- **Ephemeral** build workflows
  - Can be **deleted** at **anytime**

⏳ **Ephemeral** build environments

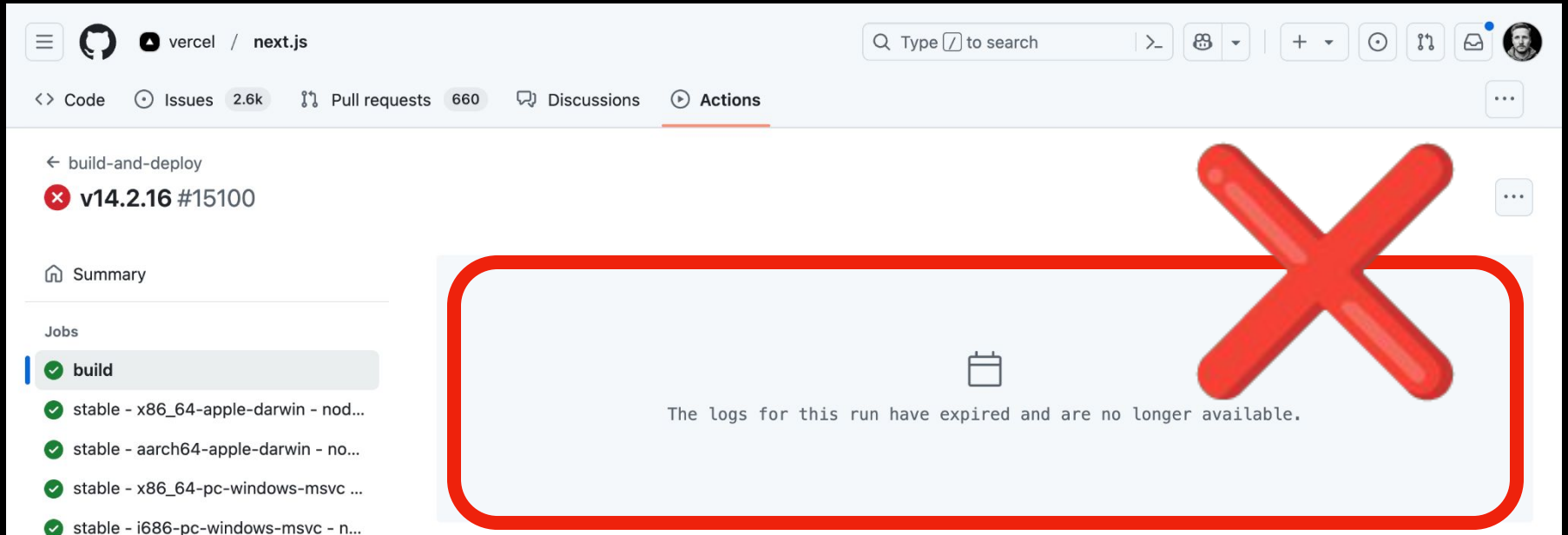
- Can be **deleted** at **anytime**
- Are **deleted** after **90 days**



producibility - @darcy



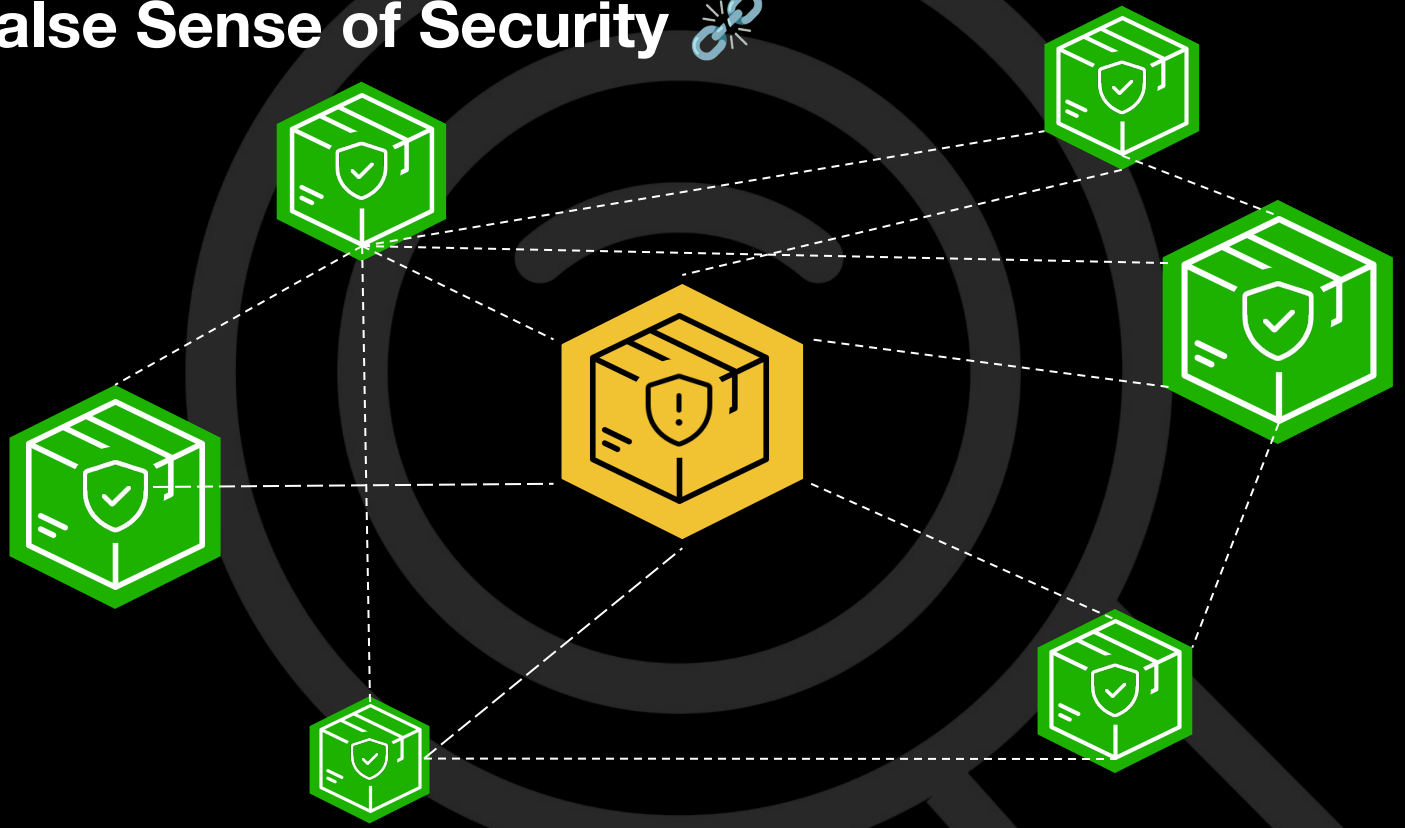
producibility - @darcy



~250k Downloads Last Week

Published **~4 months ago**

✅ False Sense of Security 🔗



What is

Reproducibility?

What is

Reproducibility?

- **Deterministic Builds**
  Running the same build process with the same inputs should always produce identical artifacts, regardless of time, location, or environment.

✅ **Verifiable Integrity** – Anyone should be able to independently rebuild a package from its source code and confirm that it matches the published version, ensuring no tampering or hidden modifications.

- **Source-to-Artifact Transparency** – The package that developers install should be probably linked to the source code it claims to originate from, reducing the risk of backdoors and supply chain attacks.

- **Independence from Trust Assumptions** – Consumers shouldn't have to blindly trust a publisher, CI system, or cryptographic signature—they should be able to verify the package contents themselves

What

Provenance & Reproducibility Don’t Do…

Tell you whether a package’s **contents** are of **good quality**

Socket

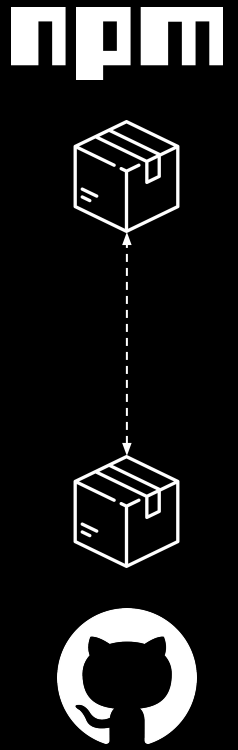
Steps to Reproduce

1. **Fetch**: package **metadata** from the “trusted” registry

2. **Setup**: published source information to…
   a. clone repository into temp dir
   b. change working directory to temp dir
   c. checkout defined ref (`gitHead` or `HEAD`)
   d. change working directory to defined `directory` or fallback to `.`

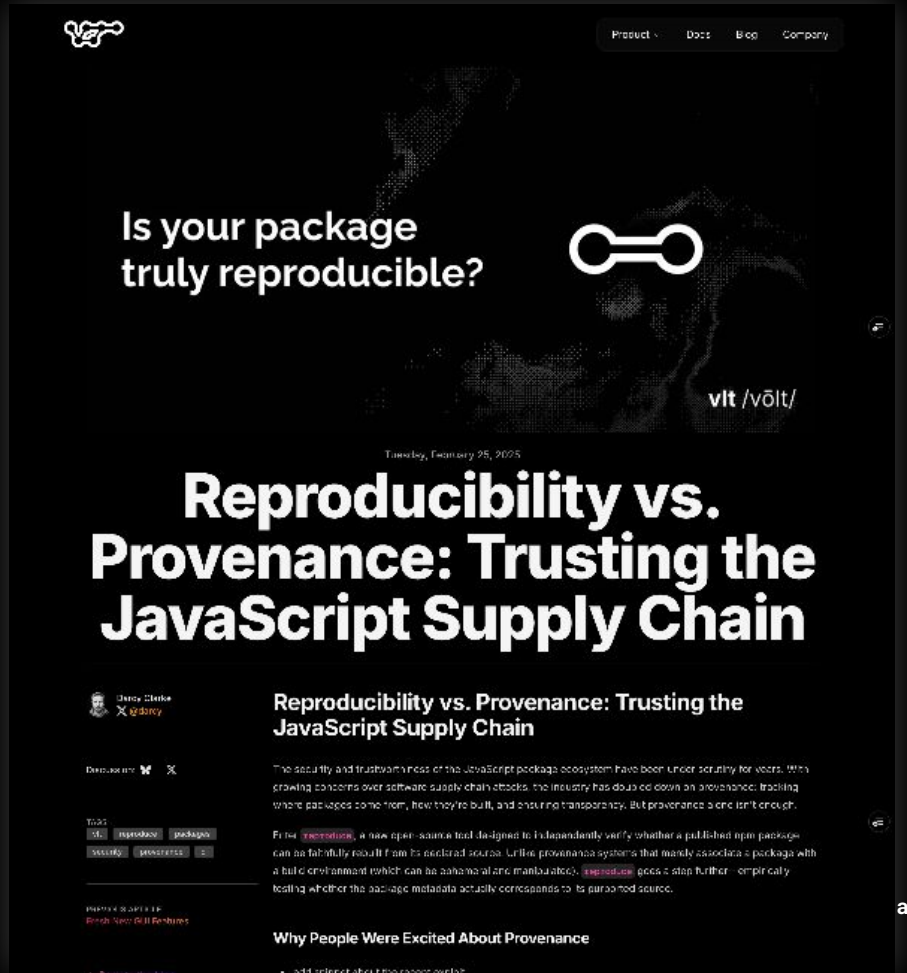
3. **Run**: a predefined strategy of build steps, ex. `npm:*`:
   a. npm install --no-audit --no-fund --silent
   b. npm pack --dry-run --json

4. **Validate**: integrity of the built package against the actual integrity



What did we

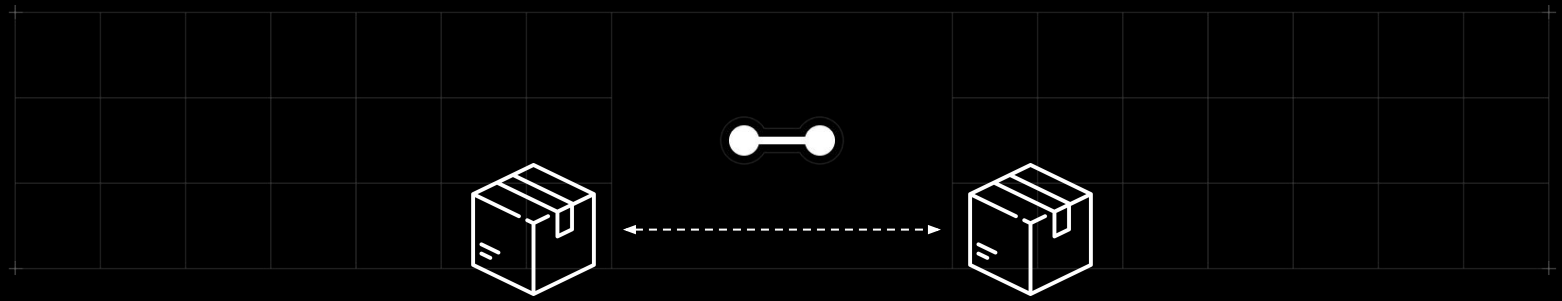
Launch?



arcy

Introducing **reproduce**

```
$ npx reproduce <pkg>
```

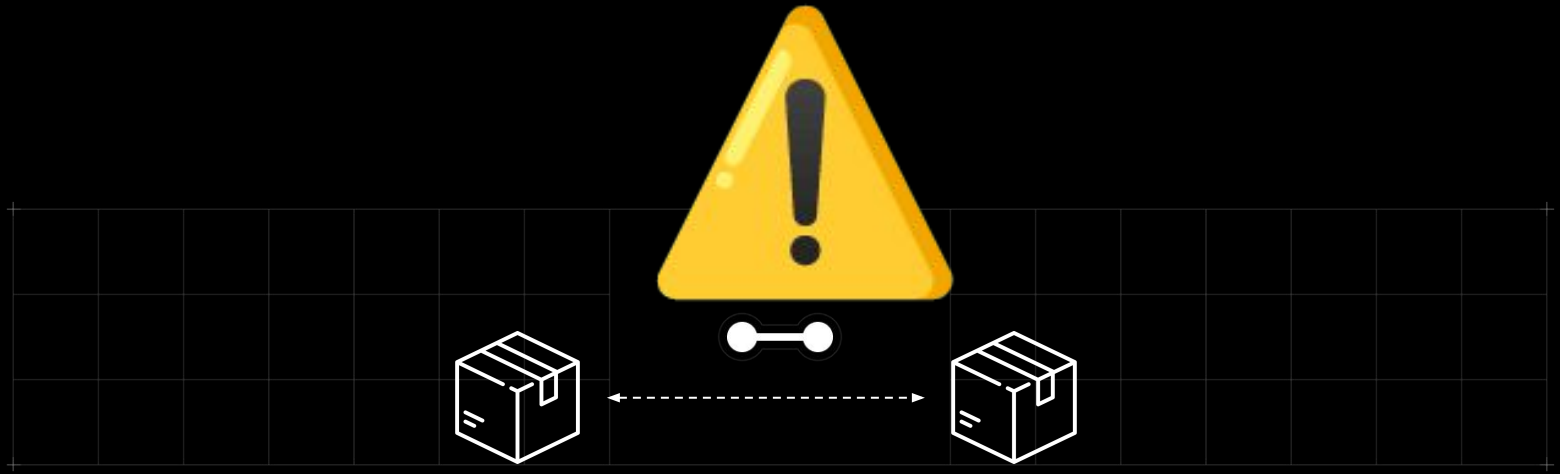


reproduce - Programmatic Usage

```
import reproduce from 'reproduce'

// Basic usage
const result = await reproduce('package-name')

// With custom configuration
const result = await reproduce('package-name', {
  cache: {},
  cacheDir: './custom-cache'
  cacheFile: 'custom-cache.json'
})
```

reproduce - CLI

```
$ npx reproduce semver
# exit code 0 - reproducible

$ npx reproduce esbuild
# exit code 1 - not reproducible

$ npx reproduce semver --json
{
  "reproduceVersion": "0.0.1-pre.1",
  "timestamp": "2025-02-25T11:08:01.729Z",
  "os": "darwin",
  "arch": "arm64",
  "strategy": "npm:10.9.1",
  "reproduced": true,
  "package": {
    "spec": "semver",
    "location": "https://registry.npmjs.org/semver/-/semver-7.7.1.tgz",
    "integrity":
"sha512-hlq8tAfn0m/61p4BVRcPzIGr6LKiMwo4VM6dGi6pt4qcRkmNzTcWq6eCEjEh+qXjkMDvPlOFFSGwQjoEa
6gyMA=="
```



Only supports **1 strategy (npm)** today but more will be added in the future (feel free to make a PR).

What

Does the data look like?

Scanned “*High Impact*” Packages (**5,000**)

>1 Million Weekly Downloads or **>500** dependants

5.78% are **reproducible**

vs.

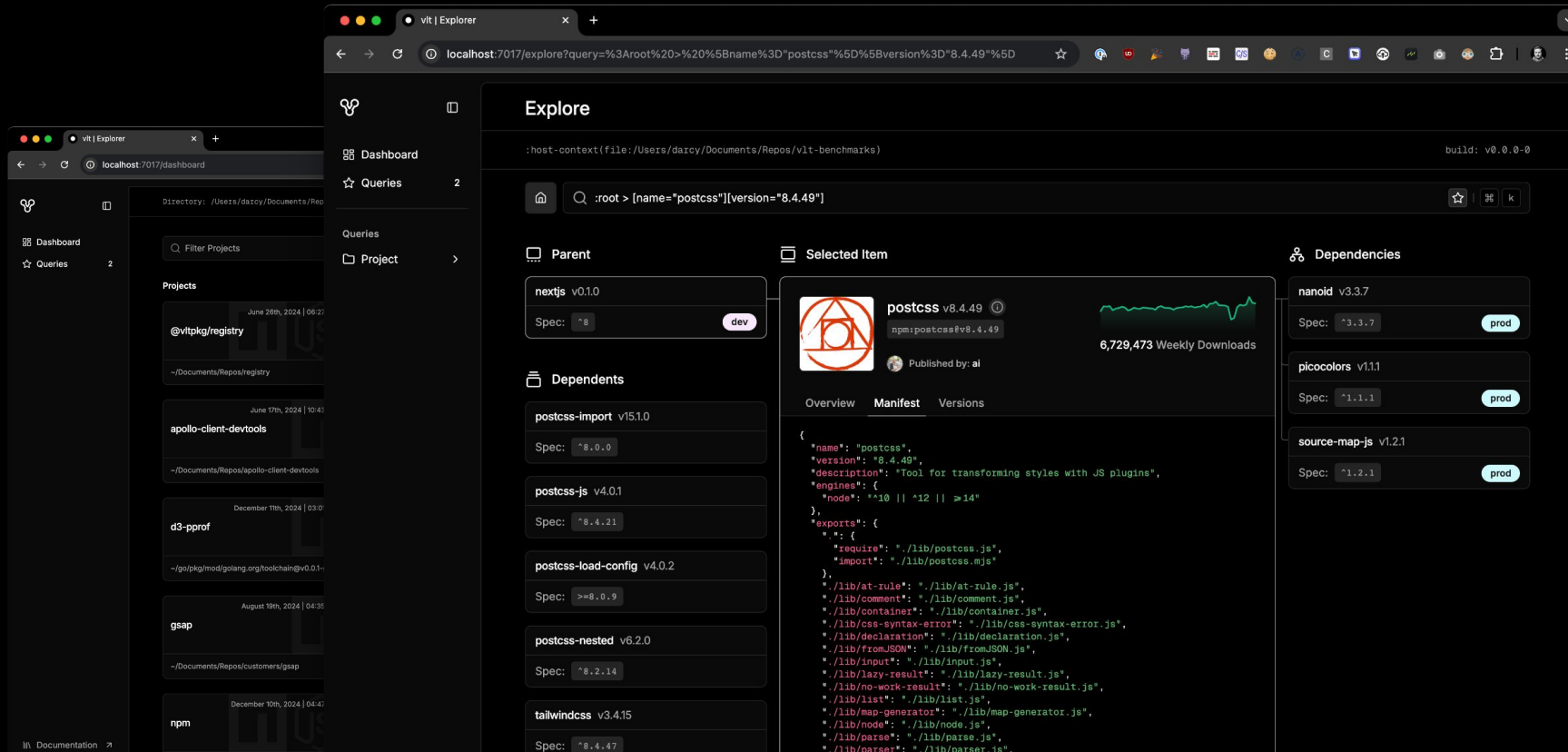
3.72% have **provenance**

Key Takeaways

- **Reproducibility >** Provenance
  - Reproducibility is independently verifiable
  - Provenance relies on **more** trust, not **proof**
- **Neither** solves for **quality**
  - Look into static analysis/malware detection tools (ex. Socket.dev)
- Try to reproduce your favorite npm package today:
  $ npx reproduce <pkg>

Next…

Modernizing JavaScript Supply Chain Security
Tomorrow @ 9:00am (same room)



Thank you!

Bluesky:
@darcyclarke.me

Twitter / X:
@darcy

GitHub:
@darcyclarke

Website:
darcyclarke.me

Bluesky:
@vltpkg.sh

Twitter / X:
@vltpkg

GitHub:
@vltpkg

Website:
vlt.sh

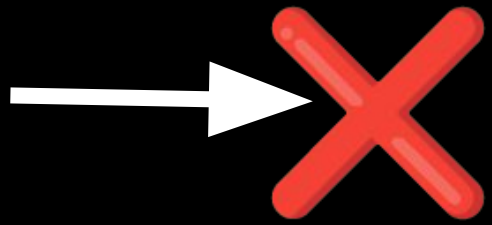
Please give feedback!

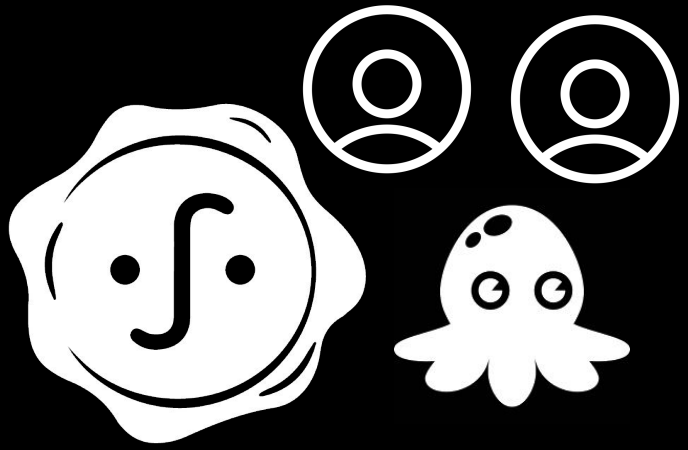
“No man ever steps in the **same river twice.**”

Heraclitus of Ephesus (Greek Philosopher)
https://en.wikipedia.org/wiki/Heraclitus

Google

Let's Encrypt





01/2021 10/2021

CLOUD NATIVE COMPUTING FOUNDATION

OPEN CONTAINER INITIATIVE

OpenSSF
OPEN SOURCE SECURITY FOUNDATION

solarwinds

THE LINUX FOUNDATION

GDPR