Supply Chain Security in the Rust Ecosystem

A case study

Alexis Mousset

Pass the SALT

4th July 2023

Who am I?

- System Lead Developer @ rudder.io
  - Open-Source Infrastructure management
  - Rust components

- Member @ Rust Secure Code Working Group
  - Vulnerabilities database for Rust libraries
  - Security-related tooling & docs

What is Rust?

- System programming language
  - C & C++ space
  - *“Memory safety without a garbage collector”*
- Compiled language
  - LLVM toolchain, performance on-par with C & C++
  - Static compilation, no stable Rust ABI
- Relatively young
  - Started in 2006, 1.0 released in 2015

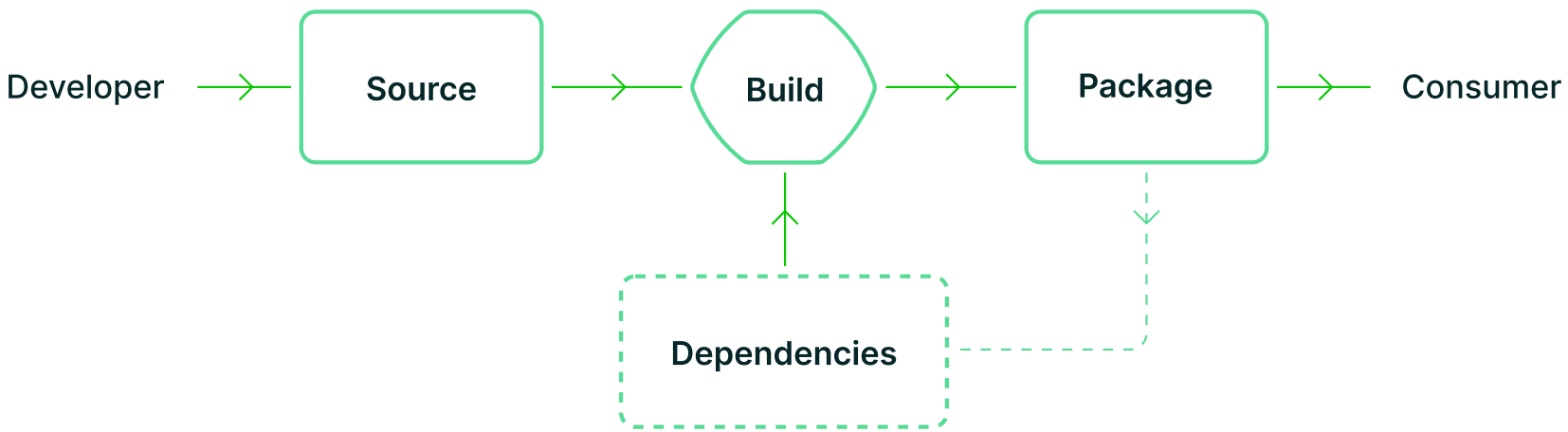
Rust for Security?

- Security was not the primary motivation
  - But a notable factor in Rust success
  - Goes beyond memory safety (type system, thread safety, etc...)

cargo

- **`cargo`** package manager
  - handles all user interaction
- A Rust package is a **`crate`**
- **`crates.io`**: public repository
  - 119k crates (2m in **`npm`**, 464k in **`pip`**)
- “Dependencies-oriented” language

Software supply chain?



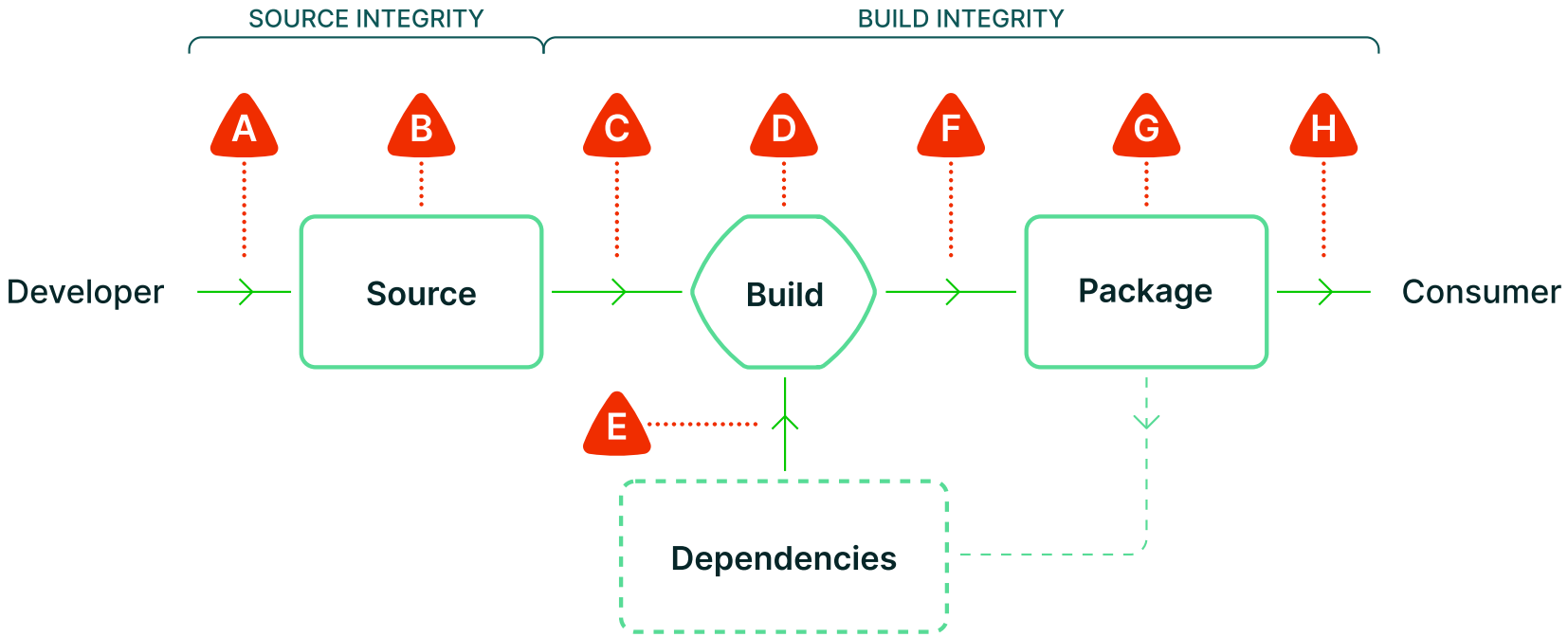
(SLSA project, under *Community Specification License 1.0*)

Software supply chain security?

Attack upstream

SLSA, OpenSSF, SPDX, SBOM, CSAF, VEX, SCA, SSDF, GUAC, GitBOM, ADG, OmniBOR, CycloneDX, SWID, Cosign, Alpha-Omega, CoSWID, OSV, SAST, SAF, OpenVEX, SaaSBOM, VDR, Rekor, TUF, SCIM, SDLC, CPE, OSS-SSC/S2C2F, DAST, purl, Fulcio, in-toto, SSCP, CVE, EO 14028, FRSCA, CBOM, SWHID, VSA, CVRF, *etc.*

Acronyms. Acronyms EVERYWHERE.



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

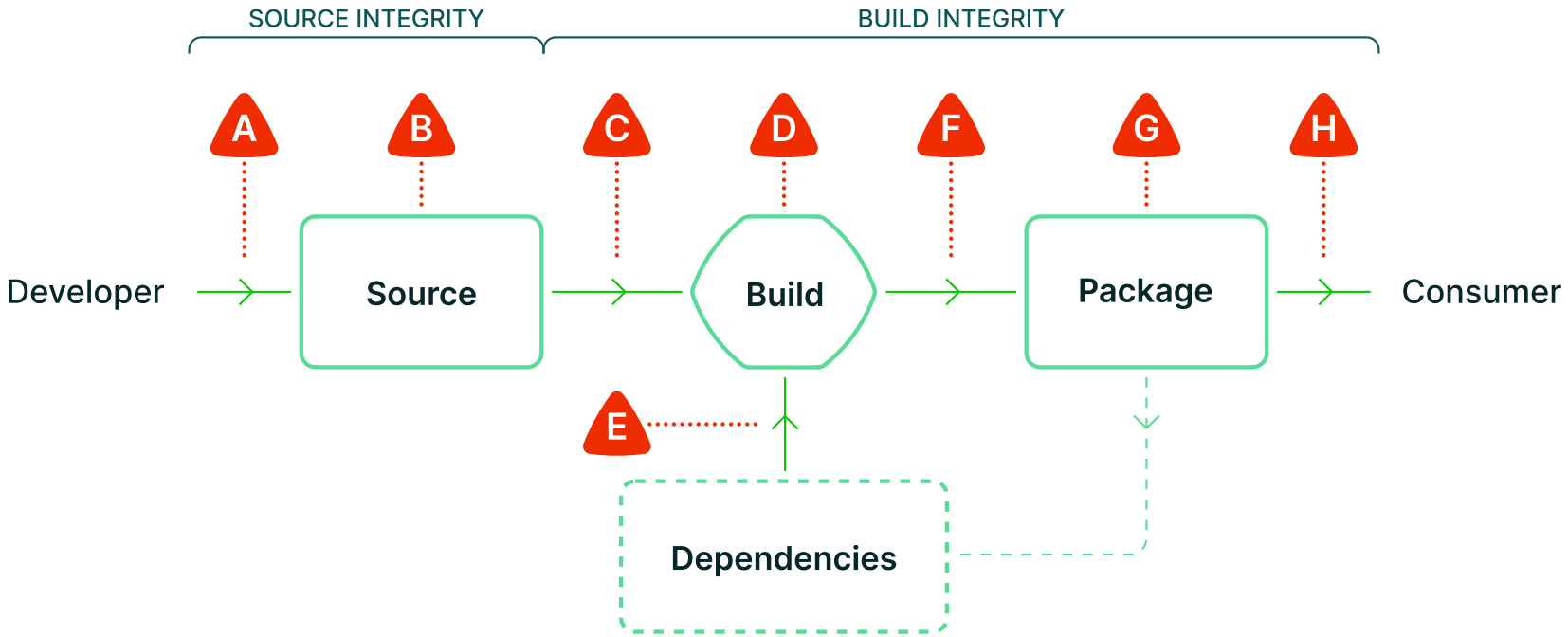
H Use compromised package

(SLSA project, under *Community Specification License 1.0*)

Source

The developer

- A workstation with a pile of software
- Various credentials
- Ex: CircleCI attack (Jan. 2023)
  - Malware on a engineer's laptop to steal an SSO session



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

(SLSA project, under *Community Specification License 1.0*)

Dependencies

i.e. the other developers.

A lot of them.

Dependencies

- Who has (indirect) push access to software?
  - Everyone that has push and release access to all your dependencies
- More and more languages, package managers and dependencies sources
  - Less reliance on system dependencies

cargo-supply-chain

- First step is visibility
  - **`cargo-supply-chain`** project (not built-in)

cargo supply-chain

- A small network daemon in Rudder
  - **`rudder-relayd`** (http server, postgreSQL access, cryptography, async runtime)
  - 240 dependencies
- 139 individuals and 34 GitHub teams.
  - hundreds of individuals
  - write access to our software
- It **is** a problem
  - We can’t just “stop using dependencies”

How hard is an attack?

So what?

- **Good**: People are generally nice to each other!
- **Bad**: It’s basically our only protection

Malicious crates

- Rust dependencies can run arbitrary code easily
  - Even by just loading then in an editor (**`proc-macros`**)
  - Arbitrary build scripts (**`build.rs`**)
- All classic “central package repositories” niceties
  - Typo-squatting
    - Already happened, with a payload target GitLab CI
  - Take control of an existing crate

Auditing crates

- You can’t audit everything on your own
- How to make it a collective effort?
  - **`cargo-crev`**
  - **`cargo-vet`**

cargo vet

- Simple model
- Review crates and store the result in your repository
  - Support relative audits (i.e. only the diff)
  - Check in CI
- Allows sharing audits
  - Central index of audit sources
  - Includes Mozilla, Google, IRSG, etc.
- User friendly UX
  - Opens a diff in browser
  - Suggests commands

Vulnerability management

- Log4shell?
- Rust has standard answers (Go/npm/...-like)
  - A vulnerability database
    - RustSec
  - Dedicated audit tooling
    - **`cargo-audit`**
    - **`cargo-deny`**
  - Still no granularity for functions (present in advisories but not audit tools)

Focus: security advisories in open-source ecosystems

- CVEs are quite unfit for language ecosystems
- not good for automated treatment (CPE is insufficient for identification)
- reviewed by non-specialists
  - qualification is often not good
- CVSS is meaningless for libraries

Focus: security advisories in open-source ecosystems

- Automated tooling using it makes it worse
- Weaponized to force a maintainer fix a bug
  - NVD -> GHSA automated import

Focus: security advisories in open-source ecosystems

- GitHub Advisory Database
  - Good on first sight
    - User-friendly tooling (reporting, dependabot, etc.)
  - Lock-in
    - Owned GHSA ids
    - And tooling owned by GitHub

Side note: security advisories in open-source ecosystems

- Vulnerability review and qualification is better done
  - Inside the community
  - In sync with the maintainers (as much as possible)
- OSV format
  - Simpler than upper-level stuff (CSAF, etc.)
  - Sensible package identification (using **`purl`** + precise version matching)
- **`osv.dev`** database
  - syndicates each project's database
  - feeds generic auditing tools

OSV

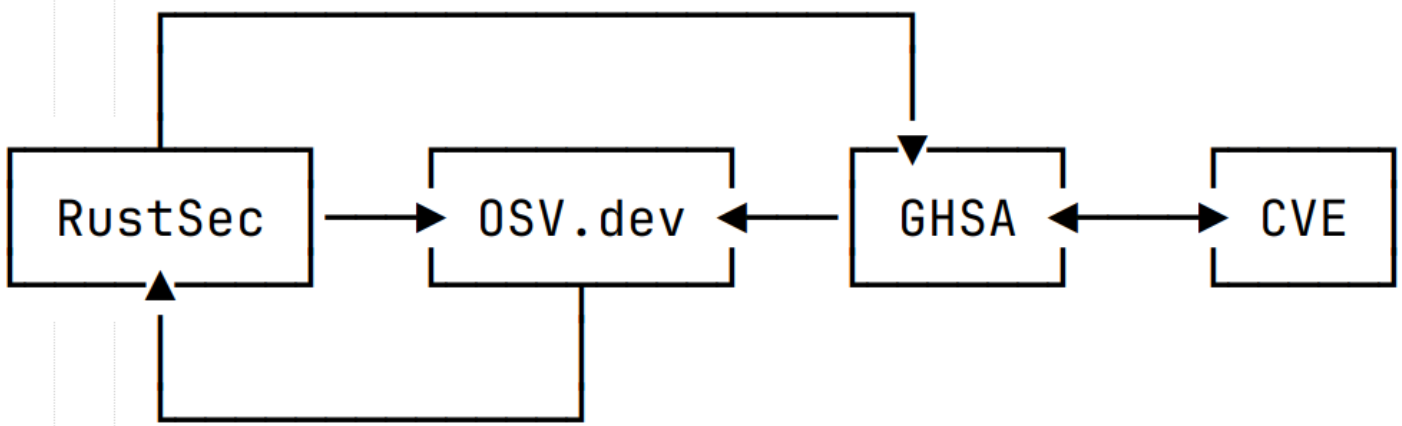
```
"package": {
  "purl": "pkg:cargo/trust-dns-server" },

"ranges": {
  "type": "SEMVER",
  "events": [{
    "introduced": "0.0.0-0" },{
    "fixed": "0.22.1" }]}
```

purl

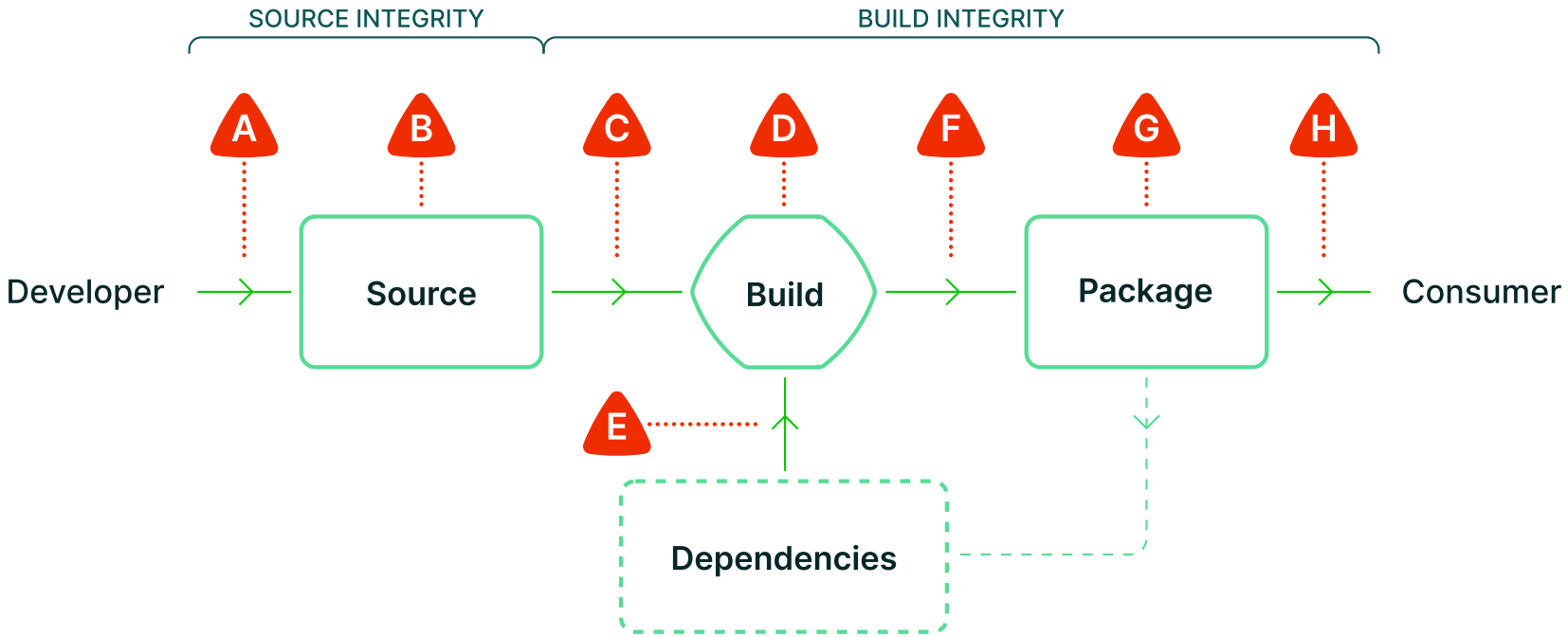
```
pkg:deb/debian/curl@7.50.3-1?arch=i386&distro=jessie
pkg:docker/cassandra@sha256:244fd47e07d1004f0aed9c
pkg:gem/ruby-advisory-db-check@0.12.4
pkg:github/package-url/purl-spec@244fd47e07d1004f0aed9c
pkg:golang/google.golang.org/genproto#googleapis/api/
annotations
```

Advisory flows



Special case: Mixed languages

- Some Rust crate embed C libraries
  - For convenience
- Used instead of the system one (openssl, gzip, etc.)
- Usually totally invisible for Rust-based tooling



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

Build

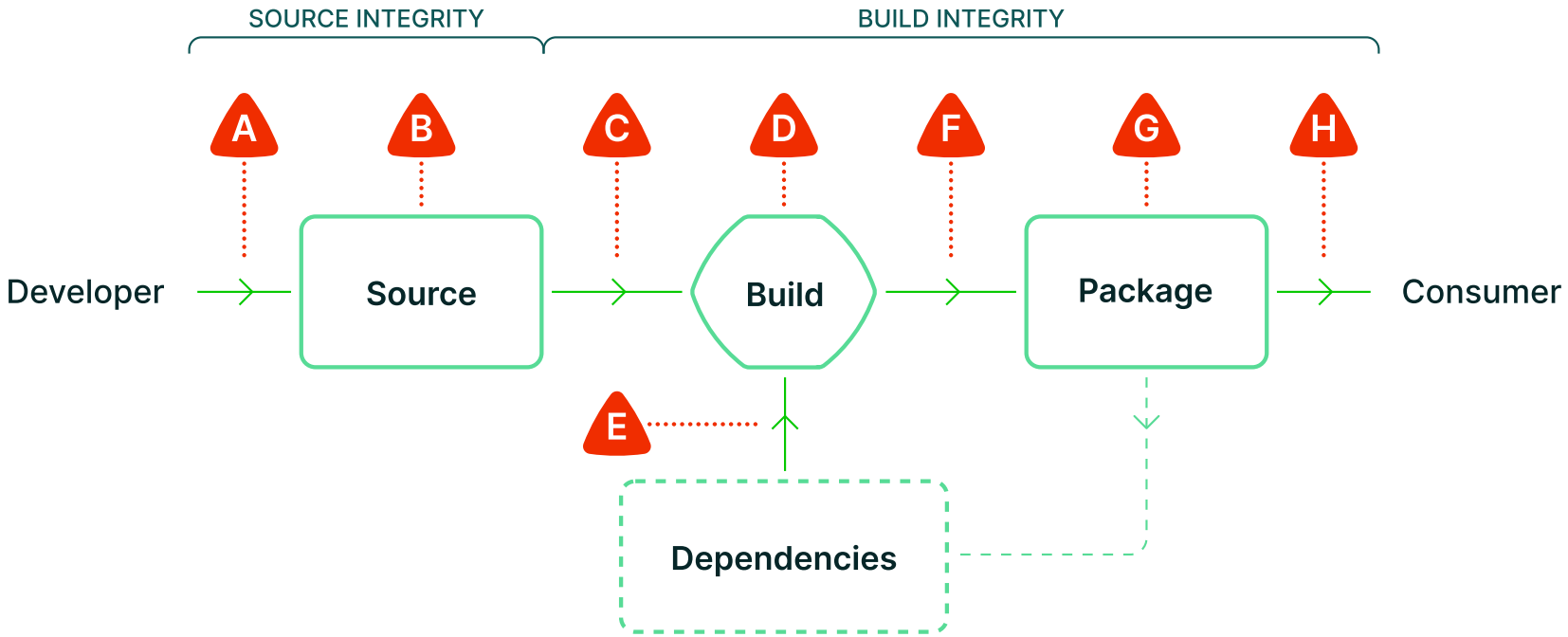
Solarwinds?

CI & Build is prod

Deterministic build envs

Build

- Hashes in lock file (**`Cargo.lock`**) in repository
  - But no transparency log
- **`cargo-auditable`**: embeds dependency list in binary
  - make the binary file auditable (**`cargo-audit`**, **`trivy`**, **`syft`**)
- *Reproducible builds* are possible but not straightforward
- SBOMs in SPDX or CycloneDX



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

(SLSA project, under *Community Specification License 1.0*)

Package distribution

Transparency? *BOM

Trust? GPG, SLSA, Sigstore

Is Rust supply chain secure yet?

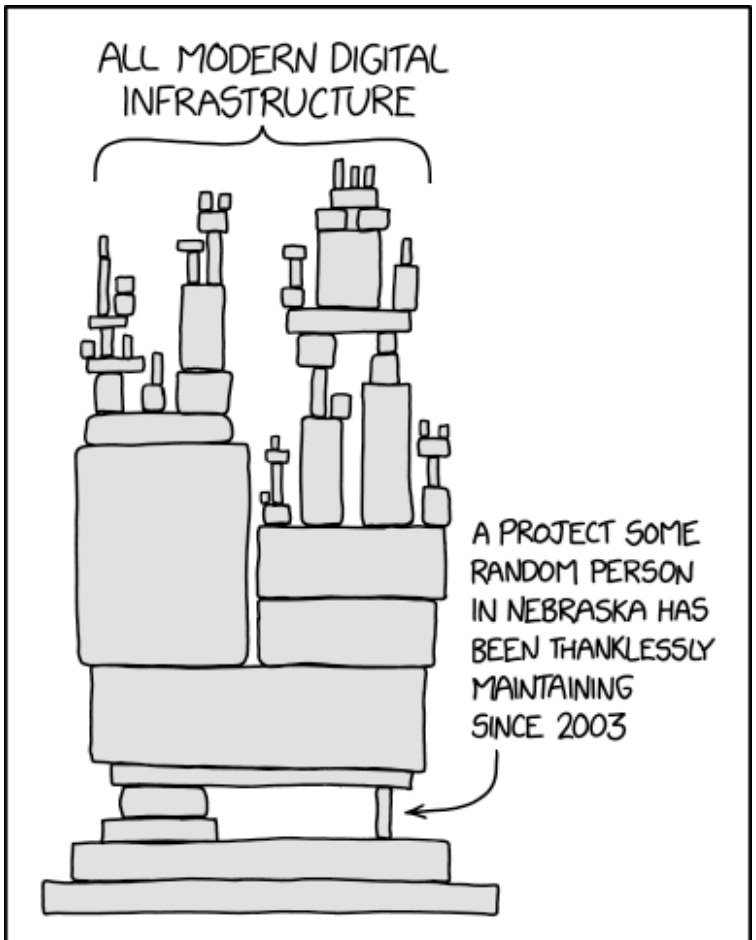
- Disclaimer: Personal opinion here
- The Rust ecosystem is not very security-aware (for non-code stuff)
- Lack of official support
  - Integration in official tools (**`cargo`** and **`crates.io`**)
- Recent improvements (thanks to the foundation and OpenSSF)
  - Optimistic for the future
- My areas of contribution: vulnerability management, import advisories from GHSA and documentation for developers
- Comparison with other ecosystems?

At Rudder

- Vulnerability monitoring is okay-ish
  - Daily audit for vulnerabilities
  - **`cargo vet`** to audit dependencies
- (pretty) Deterministic build
- No production SBOM now
- Internal CI platform

Closing words

- Supply chain security is still immature
  - Things will settle down
- Huge problem space, risk management and trade-offs
  - *“There is no secure supply chain”*
- Drowning in alerts / advisories with low added value
- Discrepancy between legal and actual security practices
- OpenSSF work is good for open-source contexts



(XKCD2347, "Dependency")

Closing words

- A problem for free software
  - We can't just make a random person in Nebraska do the security work for us
    - *E.g.: Pushback for 2FA in PyPI*
  - Legal threats (EU's Cyber Resilience Act)
  - Are we all software providers?
- Lock-in/monopoly risks (certified infrastructure for builds, GitHub Advisories, etc.)

Thank you!

amo@rudder.io

twitter.com/AlexisMousset

@amousset@mastodon.social