Do you know... 

... WHO WROTE YOUR SOFTWARE?

Marc Schoolderman
Tweede golf

May 6th, 2024
SOFTWARE MUST BECOME SAFER

- Fewer vulnerabilities
- A more reliable Internet
- Resilient critical infrastructure
Who wrote *sudo-rs*?

**Our Team**
- Developers: 🧑💻🧑💻🧑💻🧑💻
Who wrote sudo-rs?

Our Team

- Developers: 👨💻👨💻👨💻👨💻
- Outside contributors: 👤👋.depends.on.context 👩🎤
Who wrote sudo-rs?

**OUR TEAM**
- Developers: 🧑💻🧑💻🧑💻🧑💻
- Outside contributors: 🙋‍♂️‍♂️‍♂️‍♂️‍♂️‍♂️‍♂️‍♂️‍♂️‍♂️‍♂️‍♂️‍♂️‍♂️‍♂️‍♂️‍♂️‍♂️‍♂️‍♂️‍♂️‍♂️‍♂️‍♂️‍♂️‍♂️‍♂️‍♂️‍♂️

**NOT OUR TEAM**
- 11 Developers
- 1 Bot
- 4 GitHub teams
Who wrote sudo-rs?

The following individuals can publish updates for your dependencies:

1. alexcritchton via crates: glob, libc, log
2. huonw via crates: glob, libc, log
3. rust-lang-owner via crates: glob, libc, log
4. JohnTitor via crates: libc
5. KodrAus via crates: log
6. gnz1bg via crates: libc
7. joshtriplett via crates: libc
8. sfackler via crates: log

$ cargo supply-chain publishers
How many lines of code is sudo-rs?

Our team

- sudo-rs: 20.320 lines

Not our team

- log: 5594 lines
- glob: 2160 lines
- libc: 121.914 lines (bindings)

$ cargo vet
How much of `sudo-rs` is our work?

**CONCLUSIONS:**

- **At least** 13.5% of the lines of active code
- **At most** 33% of the people involved
- `sudo-rs` has **minimal** dependencies (best-case scenario)
Running example: pet project “cargo pulse”

ME:
- 1 contributor
- 181 lines of Rust
- 7 dependencies

NOT ME:
- 99 contributors, 28 teams
- 4.2 million lines of code, 1.2 million not audited
- 194 indirect dependencies
<table>
<thead>
<tr>
<th>Burden Problem</th>
<th>Trust Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="Image" alt="Burden Problem" /></td>
<td><img src="Image" alt="Trust Problem" /></td>
</tr>
</tbody>
</table>

Images: Lars Meiertoberens & kholifah, the Noun Project, CC-BY 3.0
Burden: Version management

- Easy-to-Package?

```bash
$ cargo debstatus
```

- cargo-pulse v0.1.0 (/Users/squell/cargo-pulse)
  - async-trait v0.1.80 (0.1.77 in debian)
  - cargo_metadata v0.15.4 (in debian)
  - chrono v0.4.38 (0.4.31 in debian)
  - colored v2.1.0 (in debian)
  - crates_io_api v0.8.2
    - chrono v0.4.38 (0.4.31 in debian)
    - futures v0.3.30 (in debian)
    - request v0.11.27 (0.11.24 in debian)
    - serde v1.0.199 (1.0.195 in debian)
    - serde_derive v1.0.199 (1.0.195 in debian)
    - serde_json v1.0.116 (1.0.111 in debian)
    - serde_path_to_error v0.1.16 (0.1.9 in debian)
    - tokio v1.37.0 (1.35.1 in debian)
    - url v2.5.0 (in debian)
- octocrab v0.31.2 (in debian)
- tokio v1.37.0 (1.35.1 in debian)
**Burden:** Version management

- **Duplicate, Incompatible Versions**
- **Cargo.toml up to date?**
  - (Compatibility with other versions)
- **Cargo.lock usually not included**
  - (Versions known to work)

$ cargo tree -d
Burden: License management

- Publish as Apache-2.0-OR-MIT?
- Distribute binaries?
- Respect copyleft licenses!

$ cargo license
Burden Problem

Trust Problem

(tweede golf)
Trust: Code Quality

- Undefined Behaviour (UB) impossible in normal Rust
- `unsafe` Rust: “trust me, I’m a real programmer”
- Pet project: 181 lines of safe Rust
  
  ... but 20’000+ `unsafe` expressions under the hood

$ cargo geiger
Trust: Vulnerabilities

- We know how to solve this!
- But do all developers:
  - report vulnerabilities?
  - admit embarrassing mistakes?
  - actively update versions?

$ cargo audit, cargo deny
Trust: Build time security

- Dependencies come with build scripts, **not sandboxed**!
  
  ➡ build.rs

```bash
$ cargo build
  Compiling cargo-pulse v0.1.0 (/home/cargo-pulse)
  Compiling proc-macro2 v1.0.69
  Compiling unicode-ident v1.0.12
  Compiling libc v0.2.149
  Compiling backdoor v0.1.0
[sudo: authenticate] Password:
```

- Known problem in PyPI (Python) and npm (Node.js) repositories

  [link to webinar](https://jfrog.com/webinar/identifying-and-avoiding-malicious-packages-2/)
Trust: Who is our Source of Truth?

- Do we know “Josh Triplett”, “John Titor”, “gnz1bg”, … ?
  - How do we know we are getting our code from them?
  - Can we trust them to protect their credentials?
  - Do they respond to incidents?
  - Can they be coerced to do something?
Trust: No proper authentication for dependencies

- **SSL** certificates only authenticate the package repository
- **Signed commits** are weakly authenticated (“this is my SSH key”)
- **OpenPGP** unpopular, “Web of Trust” is broken
- Linux Foundation project: [https://trustoverip.org/](https://trustoverip.org/)
What to do?

What not to do?
This is not a problem we can “fix”

- Modern software is complex

- *Many-and-small* dependencies
  - Large “Software Bill of Materials”
  - Allows analysis and risk management

- *Few-but-large* dependencies, big standard library
  - Software bloat
  - Hard to change bad design choices
Bad solution: avoiding Rust or Open source

- “Trust problem” is universal, Rust helps keep it under control
- Proprietary software **obscures** problems
Bad solution: duplicating code

- Locks you out of bug fixes
- No vulnerability reporting
- Loses licensing information
Low hanging fruit

- Critically evaluate need for dependencies, check crates.io statistics
- Give your users a choice (feature flags)
- Participate in RUSTSEC
- Work towards a standard set of “common dependencies”
- Learn from Linux distributions: reproducible builds, hygiene checks, …

- [https://www.memory­safety.org/blog/reducing-dependencies-in-sudo/](https://www.memory­safety.org/blog/reducing-dependencies-in-sudo/)
Security is a trade-off
Security is a trade-off
THANK YOU

More information available at: https://tweedegolf.nl/en/blog