

Why memory safety is not enough

# Lessons from rewriting systems software in Rust

Ruben Nijveld



# About me

---

## Ruben Nijveld

- Working at tweede golf since 2011
- Learned about Rust around 2013
- First commercial usage in 2017 (mapserver bindings)



# Trifecta Tech Foundation

---

- Januari 2022: NLNet Foundation grant for implementing a PTP prototype
- April 2022: Contracted by Prossimo (ISRG) to implement NTP in Rust
- December 2022: Contracted by Prossimo (ISRG) to implement Sudo in Rust
- August 2023: Sovereign Tech Fund (now Sovereign Tech Agency) grant for Pendulum (NTP + PTP)
- April 2024: Trifecta Tech Foundation started



**Sovereign Tech  
Agency**

# Current Trifecta projects

---

- **Time synchronization:** Pendulum (ntpd-rs and statime)
- **Privilege boundary:** sudo-rs
- **Data compression:** zlib-rs and bzip2-rs
- **Smart grid protocols:** openlead-rs
- *Education - teach-rs*
- *Making Rust faster than C*



*Open infrastructure software in the public interest*

# What is systems software?

---

“System software is software designed to provide a platform for other software”

- Generally relatively low-level software
- Relatively low overhead and high performance
- Protocols, algorithms and formats

# Why these projects?

---

## **It has to be interesting**

We're in it for the long haul, and we want to keep our work interesting

# Why not?

---

**Just find the projects with the most buffer overflow issues, right?**



# Why not?

## Pick your battles, not every project needs a rewrite

- Just rewriting in Rust is not enough
- Be careful for the negatives
  - ▶ Communities can be split
  - ▶ Users don't see any changes
- Focus on the positives, those that don't join your effort aren't the enemy
- Never fear a hobby project though



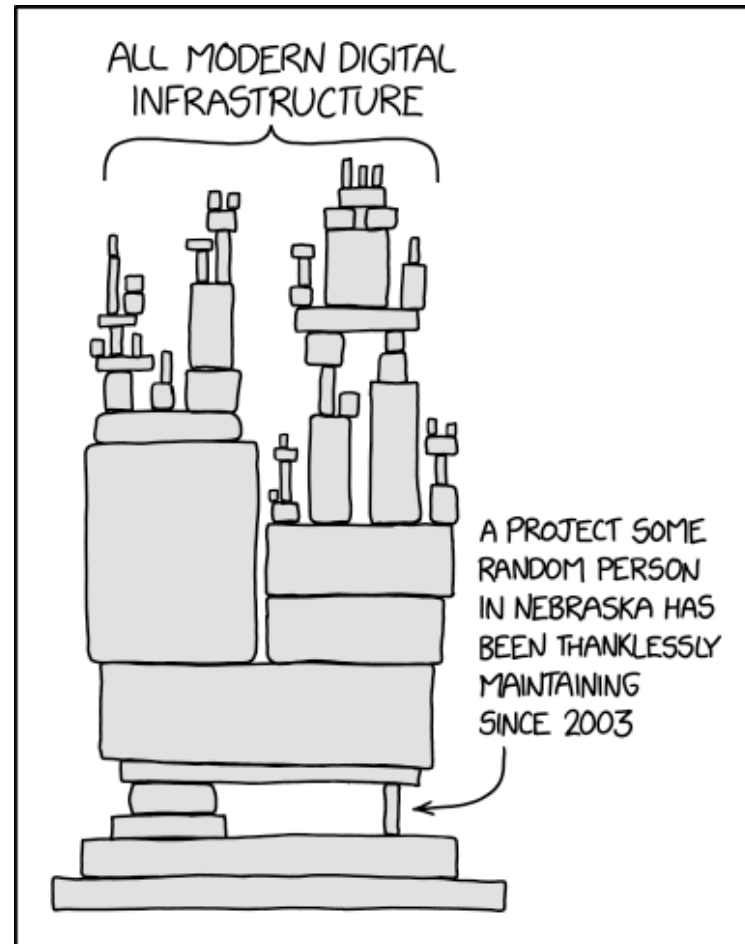


# Why these projects?

---

## Finding vulnerable projects, protocols or ideas

- Few people working on it, lots of people relying on it
- A history of security issues
- Unstructured codebase
- Not just related to the Rust ecosystem



# And now...

---

## **So we found something that could use a little Rust**

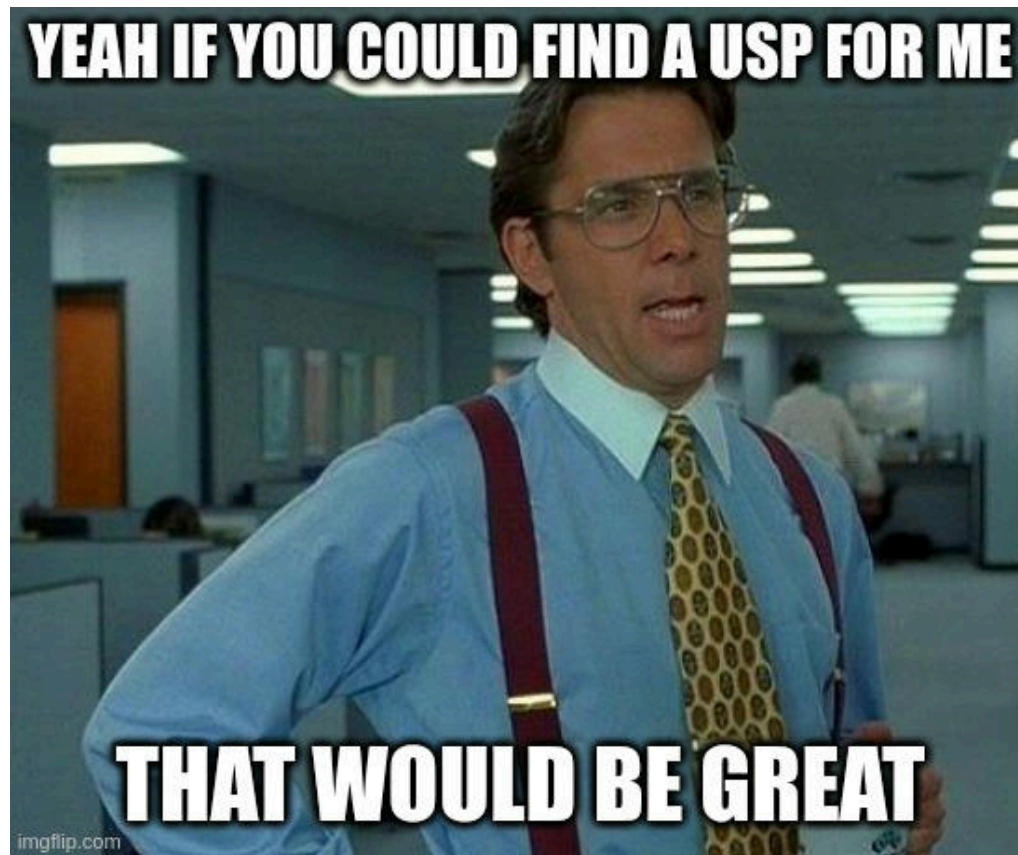
- Are you aware of a significant portion of the problem space?
- If not: just start trying stuff! Make it your hobby! Enjoy!
  - ▶ Encourage people new to the problem space, even if you don't like Rust
- Make sure to look at competing implementations (beware of licenses though)

# Your unique selling point

---

**Think about what would sell your new implementation**

- Something other than: Rust brings memory and type safety
- We Rust programmers care about type safety a lot
- This is not what your users care about though



# Focus on security

## Maybe your implementation focuses on being more secure?

- Think beyond just the memory and type safety
- In *sudo-rs*: Which features are really needed? Lower attack surface?
- In *ntpd-rs*: Focus on Network Time Security and have stricter defaults



# Focus on performance

---

## Maybe your implementation focuses on being more performant?

- Maybe a little unsafe can perform better?
- Balance security, usability and performance
- Make sure you benchmark what you are claiming
- In *zlib-rs*: Use SIMD instructions, focus on relative performance





# Focus on stability

---

## Maybe your implementation focuses on being stable in some way?

- Think about memory leaks
- Stable API: make sure your API does not need to change for a long time
- In *ntpd-rs*: Worked toward 1.0, stability guarantees
- In *zlib-rs*: Existing C zlib interface as primary way to interact



Restart  
the web worker  
every 1000  
requests to  
fix memory leaks

Write  
software  
without  
memory leaks

# Find your own focus

---

## **Your focus is your strength, and also helps others**

- Be the change you want to see
- Help other implementations, document what works and what doesn't

# Make it the best project ★ ★ ★ ★ ★

---

- From here on out, iteration is needed
  - ▶ On your code
  - ▶ On your unique selling point
- Take some time, no rush
- Some points of attention:
  - ▶ Documentation
  - ▶ Dependencies
  - ▶ Distribution



# Documentation

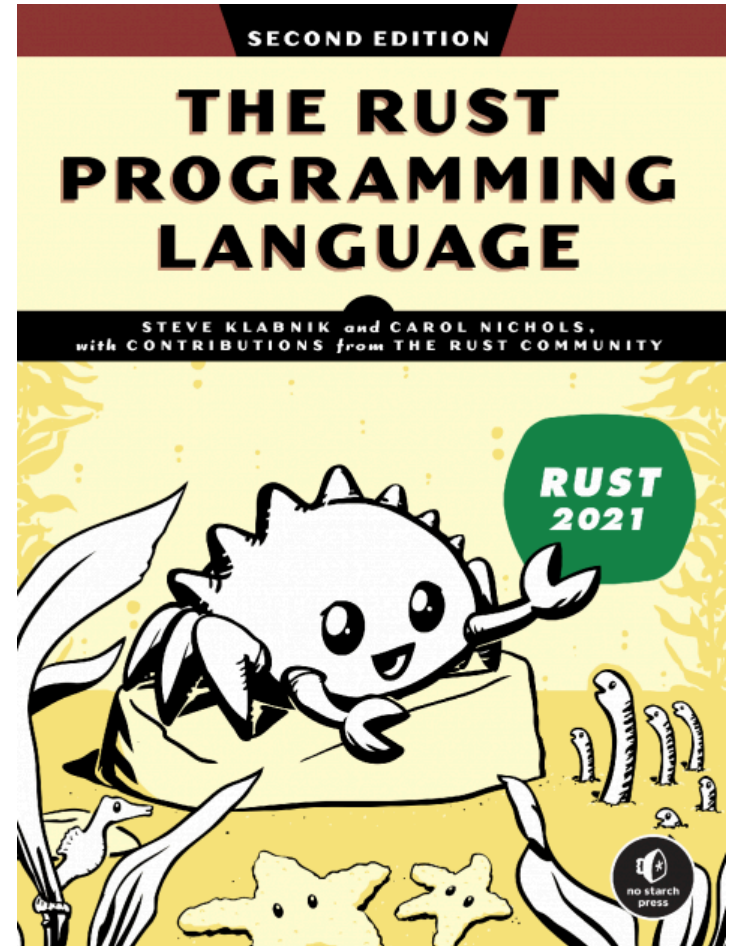
---

- As a Rust user we clearly make the best ever API documentation
- You obviously have `#![forbid(missing_docs)]` and `#![forbid(clippy::undocumented_unsafe_blocks)]`
- So every public interface is expertly document, including tested examples

# Documentation

---

- Your users know way less than you do
- Just API docs with rustdoc is not enough
- Examples and tutorials
- High level guide
- Reference guide



# Dependencies

---

- Dependencies solve problems so you don't have to
- The Rust ecosystem offers a lot of libraries these days
- New projects: use them! All of them!



# Dependencies

---

## Dependencies are a risk as well

- A burden risk
  - ▶ Don't take on more dependencies than you need
- A trust risk
  - ▶ Only take on well known dependencies with trusted maintainers
- A dependency's goals might not always align with yours
- Make sure you take these into consideration



# Dependencies

---

- Trust is nice, verify is better
- You can use tooling like `cargo vet` to share review load for dependencies
- Dependencies can also be part of your distribution story



# Dependencies and Debian packaging

---

- Targeting Linux? Debian-based distributions will be large part of your userbase
- Debian packages each of your dependencies as individual source packages
- Only one version of a package will be available in a Debian distribution

## MANAGING RUST DEPENDENCIES



# Distribution

---

- Just publish on crates.io right?
- Your users aren't Rust users, and don't need or want a Rust compiler
  - ▶ `cargo install` is not a distribution mechanism
- If your intent is to replace an existing piece of software:
  - ▶ Using your software should be just as easy as the one you are replacing
- (Linux) distributions and other downstream maintainers are your friends!
  - ▶ Rust packaging tools often prefer to use crates.io as a source
- The biggest impact can be made by supporting the widest ecosystem

# Whoops, my crate name is not available

---

- No need to get it right the first time, you can always rename
- People are horrible at naming things
- Just ask, quite often people will respond
- It helps if you have something to show



# You need to build up trust

---

- In the end a lot of what we're doing is about people
  - ▶ Users
  - ▶ Contributors
  - ▶ Dependency maintainers
  - ▶ Downstream maintainers
- Show that you are a reliable partner
- Have a way to handle security issues

# So what have we done?

---

- We picked the project that needed RIIR treatment
- We found something to focus on that uniquely identifies our project
- We iterated and created the best software package ever
- We made the best documentation ever
- We limit, trust and verify our dependencies
- We distribute our software to the widest audience
- We've communicated so well that everyone trusts us

# So what have we done?

---



Just download my software already!

# Thank you!

---

## Any questions?

**E-mail** ruben@tweedegolf.com

**Tweede golf** <https://tweedegolf.nl/>

**Trifecta** <https://trifectatech.org/>

**ntpd-rs** <https://github.com/pendulum-project/ntpd-rs/>

**sudo-rs** <https://github.com/trifectatechfoundation/sudo-rs/>

**zlib-rs** <https://github.com/trifectatechfoundation/zlib-rs/>