A Taste of Rust

The Zeuthen Data and Software Seminar 2025-02-25

Sebastian Wagner
Zeuthen, 2025-02-25

HELMHOLTZ

DESY.

What
when
... why?

Rust History

A side project – *again*!

- Started by Graydon Hoare in 2006 as a side project, sponsored by Mozilla in 2009 [1]

- Today Rust systems and components support many use cases – some examples:
  - **Desktop Software:** e.g. in Firefox [2] and Dropbox [3]
  - **Service Backend / DB:** e.g. at Discord [4] and Figma [5]
  - **Cloud Infrastructure:** e.g. at Amazon / AWS [6], Cloudflare [7]
  - **OS / Mobile:** e.g. at Google for Android [8], used in small parts of Windows [9], whatever is happening in the Linux kernel [10]
  - **Embedded:** e.g. at Oxide [11]
  - **Developer Tooling / CLI:** e.g. uv [12], ruff [13], ripgrep [14]

- **Reminder:** just because someone can pull this off does not mean it is also a good idea for you

Sources (retrieved 2025-02-19):

[1] https://www.technologyreview.com/2023/02/14/1067869/rust-worlds-fastest-growing-programming-language/

[2] https://hacks.mozilla.org/2016/07/shipping-rust-in-firefox/

[3] https://dropbox.tech/infrastructure/rewriting-the-heart-of-our-sync-engine

[4] https://discord.com/blog/how-discord-stores-trillions-of-messages

[5] https://www.figma.com/blog/rust-in-production-at-figma/

[6] https://aws.amazon.com/de/blogs/opensource/why-aws-loves-rust-and-how-wed-like-to-help/

[7] https://blog.cloudflare.com/tag/rust/

[8] https://www.youtube.com/watch?v=QrrH2lcl9ew

[9] https://www.youtube.com/watch?v=8T6ClX-y2AE&t=2977s

[10] https://rust-for-linux.com/

[11] https://github.com/oxidecomputer

[12] https://github.com/astral-sh/uv

[13] https://github.com/astral-sh/ruff

[14] https://github.com/BurntSushi/ripgrep

Why Rust?

- Rust is a compiled [1], statically typed language
- Main promises
  - **Performance:** fast execution, efficient memory management [2]
  - **Reliability:** memory and thread-safety without garbage collection, entire classes of errors caught at compile time [2]
  - **Productivity:** Good tooling, error messages, documentation, auto-completion etc. [2]
- Careful with absolutes / benchmarks :)

[1] Reference https://rust.godbolt.org/ (retrieved 2025-02-21)

[2] Source https://www.rust-lang.org/ (retrieved 2025-02-19)

Why Rust?

- Basically, ‘go as fast as C/C++ with fewer bugs’
- Core features (my selection)
    - Ownership/borrow-checking/lifetimes → the memory safety *similar* to GC without the performance penalty [1]
        - Yes, the compiler yells at you a lot in the beginning, but chances are high what you were trying to do was not a smart idea™ in the first place
        - If it compiles, there’s a decent chance it works, less crashes can be nice especially for long-running tasks
    - The basic features you expect from a ‘modern’ language
        - Structs [2], functions [3], traits [4], results [5], pattern matching [6]...
        - Abstractions + tooling allow you to both leverage low-level control and high-level abstractions which can be great for productivity (we’ll see this in a live demo)
    - Fearless concurrency [7]
        - Concurrency is only becoming more important
- Takes some getting used to, however, I think it’s worth it
- Learning Rust can change the way you think in other languages

[1] Source https://rust-book.cs.brown.edu/ch04-05-ownership-recap.html (retrieved 2025-02-24)
[2] Reference https://doc.rust-lang.org/book/ch05-01-defining-structs.html (retrieved 2025-02-24)
[3] Reference https://doc.rust-lang.org/book/ch03-03-how-functions-work.html (retrieved 2025-02-24)
[4] Reference https://doc.rust-lang.org/book/ch10-02-traits.html (retrieved 2025-02-24)
[5] Reference https://doc.rust-lang.org/book/ch09-02-recoverable-errors-with-result.html (retrieved 2025-02-24)
[6] Reference https://doc.rust-lang.org/book/ch19-01-all-the-places-for-patterns.html (retrieved 2025-02-24)
[7] Reference https://rust-book.cs.brown.edu/ch16-00-concurrency.html (retrieved 2025-02-24)

Why Rust?

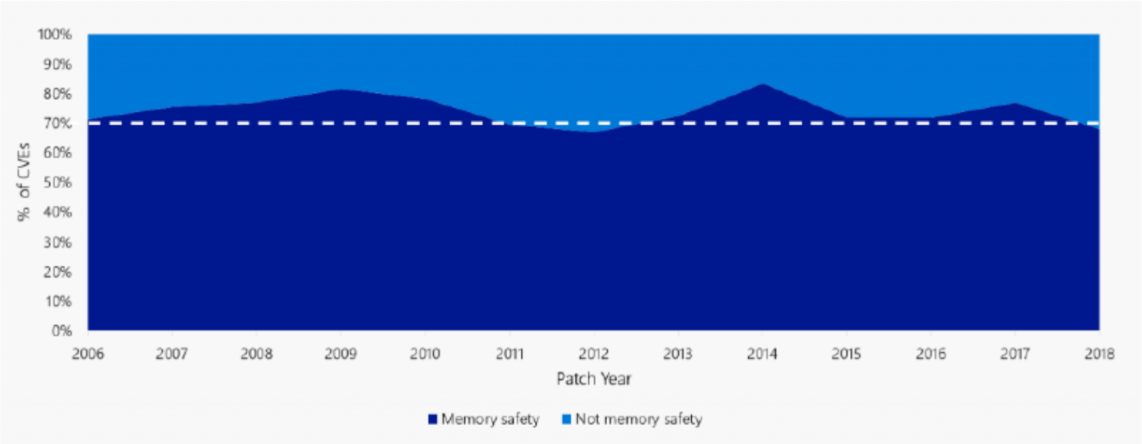
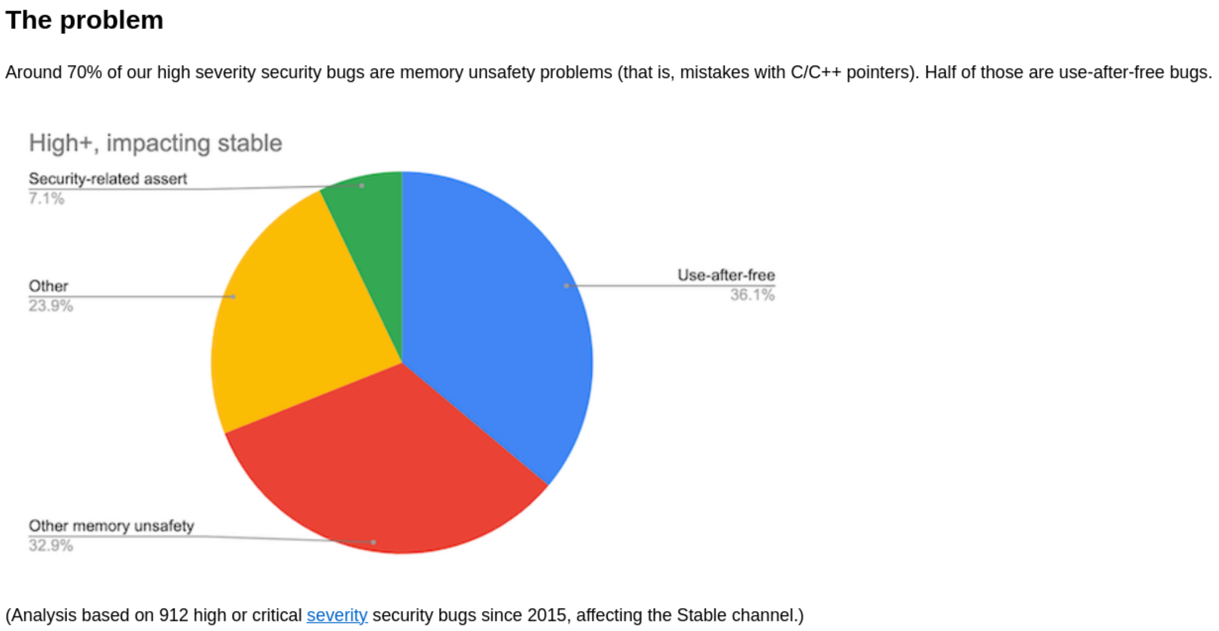


Figure 1: ~70% of the vulnerabilities Microsoft assigns a CVE each year continue to be memory safety issues

Source: Screenshot (retrieved 2025-02-19): https://msrc.microsoft.com/blog/2019/07/a-proactive-approach-to-more-secure-code/



The problem

Around 70% of our high severity security bugs are memory unsafety problems (that is, mistakes with C/C++ pointers). Half of those are use-after-free bugs.

(Analysis based on 912 high or critical severity security bugs since 2015, affecting the Stable channel.)

Source: Screenshot (retrieved 2025-02-19): https://www.chromium.org/Home/chromium-security/memory-safety/

A Taste of Rust
Mini Live Demo

The Ecosystem
and
Practical Considerations

Should I use Rust for my project?

General Considerations

Should I use Rust for my project?

This section is *based on personal experience and highly opinionated*, **do your own research**!

- **Remember:** Rust is *a* tool in your toolbox not *the* tool!
  - You can get lost in benchmarks and programming language wars but in the end what counts is whether the tool enables **you and your team** to make a positive impact (maybe even efficiently)
  - When choosing a language for your project **the ecosystem is a critical component**!
- **Easiest:** Would you write it in C/C++? → Seriously consider Rust
  - Interoperability with existing C/C++ libraries is possible, check if that's a viable option for you
- Problem properties where Rust may give you an advantage:
  - You have **a well-defined problem** you're trying to solve (hence the rewrite in Rust meme)
  - You anticipate you're going to have to tightly **control how your code interacts with the hardware**
  - Problems **where crashes would be expensive** (embedded, servers…)
  - **Systems-heavy work** e.g. lots of interacting components, large/volatile teams, concurrency etc.
  - ...

General Purpose Development

Use Case: Run-off-the-mill generic projects, one-off CLI tools etc.

- **Examples:** package ecosystem [1, 2], CLI parsing [3], TUI [4], parsing/serialization [5], simple HTTP client [6], tests [7]
- **Notes**
  - The basic ecosystem is there (see above), the standard library is decent and documented [8]
  - You may need some time to get used to the intricacies of the type system/marker types
  - Editor support is solid [9]
  - Distributing binaries is a breeze [10] compared to many other languages
- **Where to start**
  - After the basics: intro to building CLI apps in Rust: https://www.rust-lang.org/what/cli

Sources (retrieved 2025-02-21):
[1] https://crates.io/
[2] https://lib.rs/
[3] https://github.com/clap-rs/clap
[4] https://ratatui.rs/
[5] https://serde.rs/
[6] https://github.com/seanmonstar/reqwest
[7] https://doc.rust-lang.org/book/ch11-01-writing-tests.html
[8] https://doc.rust-lang.org/std/index.html
[9] https://www.rust-lang.org/learn/get-started → ‘Other Tools’
[10] https://rust-cli.github.io/book/tutorial/packaging.html

(Distributed) Systems Development

Use Case: Reliable (networked) high-throughput services e.g. controls / data acquisition / processing

- **Notes**
  - tokio + related projects [1]
  - Pretty strong ecosystem that allows you to go fast in places where you absolutely must
  - Spend some time to learn about concurrency/parallelism first
  - Depending on where you spend your CPU cycles, compared to other solutions development ergonomics may be a bit clunky in Rust (the type system gets rather intense here (why?))
    - Still worth the effort if the alternative is requiring magnitudes more hardware or time

- **Alternatives to consider:** e.g. Elixir / Erlang / BEAM… [2] for more dynamic systems, fault-tolerance and higher-level abstractions, Go [3] for the ‘cloud’ ecosystem (easier to learn)

Sources (retrieved 2025-02-21):
[1] https://github.com/tokio-rs/tokio
[2] https://elixir-lang.org/
[3] https://go.dev/

Scientific Computing

Data Acquisition / Handling / Storage / Online Processing

Use Case: You have that pile or stream of data where your other tools eat all your memory and/or CPU

- Basically another subset of the previous two, you got the toolkit for:
  - Type-safe data parsing
  - Concurrent I/O
  - Networked services
  - CLI infrastructure
  - Control over memory usage
  - Testing tools
  - There are even MPI bindings (e.g. [1]), I can’t say how good they are, not a HPC guy
- **Caveat:** your favorite math library may not be available

Sources (retrieved 2025-02-21):
[1] https://github.com/rsmpi/rsmpi

Scientific Computing

Exploratory Analysis / Language Extension

Exploratory Analysis

- Interactively fiddling with data and visualizations in a notebook-style project where you're figuring-out your problem on the fly iteratively
- Rust may not be the best fit here: it really shines if you can define your problem well and know where you want to go, visualization may technically be possible [1, 2] but may not be the best use of your time (feel free to disagree)
- e.g.: you could prototype in Python or Julia and then implement the solution in Rust once you got the algorithms right (e.g. like here [3], discussion here [4]), you can then even validate solutions against each other

Extending other Languages

- You can also extend other languages with Rust code such as Polars dataframes in Python [5] (you can also use them from Rust or how Discord extended their Elixir code with a Rust data structure [6]

Sources (retrieved 2025-02-21):

[1] https://github.com/plotters-rs/plotters
[2] https://github.com/rerun-io/rerun
[3] https://youtu.be/4mDCUHb7XLY?t=169
[4] https://discourse.julialang.org/t/blog-post-rust-vs-julia-in-scientific-computing/101711/10
[5] https://github.com/pola-rs/polars
[6] https://discord.com/blog/using-rust-to-scale-elixir-for-11-million-concurrent-users

Embedded

Use Case: Firmware for sensor boards, motor controls etc.

Notes

- Rust seems like a natural fit for this area, standard-library less environments are supported, memory management can be controlled
- Compared to C / ASM you get a bunch of useful checks at compile time
- As usual with embedded systems a lot seems to hinge on individual hardware / board support
- Future will show whether Rust is going to thrive in typical embedded systems environments such as automotive / medical / aerospace
  - Looks promising, maybe just industry certifications and the ecosystem need some time

Where to go

- Overview: https://www.rust-lang.org/what/embedded
- Embedded Rust Book: https://docs.rust-embedded.org/book/
- More resources: https://github.com/rust-embedded/awesome-embedded-rust

Where to go from here?

Where to go from here

- Either way, I recommend giving it a try
  - Worst case you spend some time and learn a bit about programming
  - Learning Rust can change the way you think in other languages
- Lots of excellent free resources available online
  - Getting started: https://fasterthanli.me/articles/a-half-hour-to-learn-rust
  - The Book: https://doc.rust-lang.org/book/
  - Exercises: https://github.com/rust-lang/rustlings/
  - Project website: https://www.rust-lang.org/
- Annual scientific computing workshop + recordings: https://scientificcomputing.rs/
- Rust Workshop @ DESY
- Talk to people and get connected

Thank you

Contact

Deutsches Elektronen-Synchrotron DESY

www.desy.de

Sebastian Wagner
FH / IT / Research & Innovation in Scientific Computing / HIFIS

sebastian.wagner@desy.de