Unleashing 🦀 The Ferris Within

NDC { TechTown }

September 2024

🐦 @ciura_victor
🦣 @ciura_victor@hachyderm.io
🦋 @ciuravictor.bsky.social

Victor Ciura
Principal Engineer
Rust Tooling @ Microsoft

Abstract

"Let's rewrite it in Rust" is no longer a party joke. It's happening!

Let me share a couple of stories of learning, appreciating and rewriting stuff in Rust. How we came to love 🦀 Ferris: cargo cult or real need?

What is it like to come to Rust from two very different directions: C++ and C#? What are the gaps, the needs, the gems and the tools you should know about? Here's a real journey and the various experiments leading up towards the success stories at Microsoft.

What have we learned and can bring back to day-to-day C++?
Want to compare notes? Let's chat.

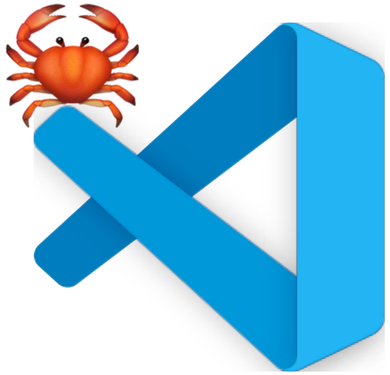
About me

Advanced Installer

Clang Power Tools

Oxidizer SDK

Visual C++



Rust Tooling

@ciura_victor

@ciura_victor@hachyderm.io

@ciuravictor.bsky.social

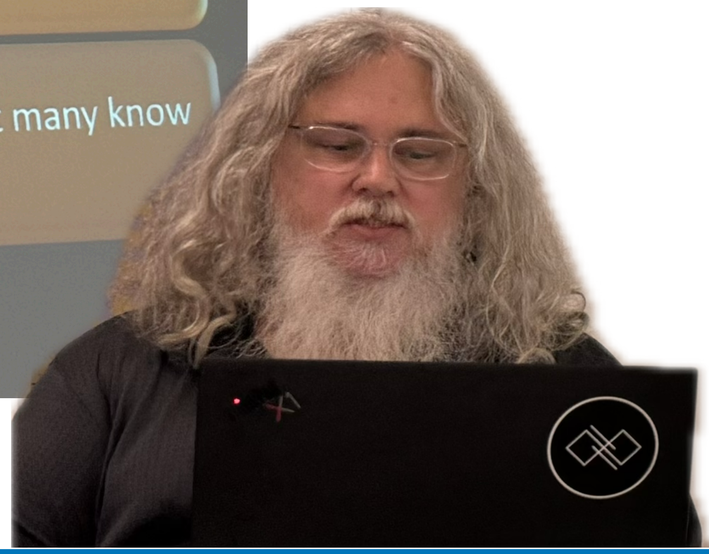
So Why Rust is Not Widely Adopted?

Mature ecosystem: Maturity of the safety ecosystem built around C and C++ (frameworks/standards/processes) vs Rust. Support for safety certified products in C and C++ is broad, lots of tools, lots of assessors, lots of companies to cover the liability, lots of standards.

Lack of tooling for Rust: Mathworks doesn't offer Rust code generation from Simulink models, for example.

Hardware support: most hardware provides C style APIs. For example, the AEM GlobalPlatform Trustzone API is C style, so even if your trusted execution environment supports Rust you would then have interop which is often buggy, so why not just do C?

Existing engineering skillset: people that work in this space know C and C++ already, but not many know Rust. New programmers add bugs; mentorship from experts required to avoid bugs.



ndctechtown.com/agenda/memory-safety-rust-vs-c

Rust ❤️ C++

I'm not here to:

- convert anyone to 🦀 Rust
- start any language wars
- *"sell the Rust snake oil"*
- tell you to RiiR

So, don't throw 🍅

Rust ❤️ C++

Engineering

Engineering, not programming

“Software engineering is programming integrated over time.”

abseil.io/swe-book/ch01

Engineering

Bootstrapping a team/project

Finding the right combination of skills (systems, services, Rust)

Expectations, misconceptions, myths

Path... 🦀

- Why teams want Rust
- Path to Rust
  - Learning
  - Bootstrapping
  - Engineering Systems
  - RiiR
- Interop aka. "not living in a bubble"
- Problems along the way
- Early wins

But Why?

But Why?

But Why?

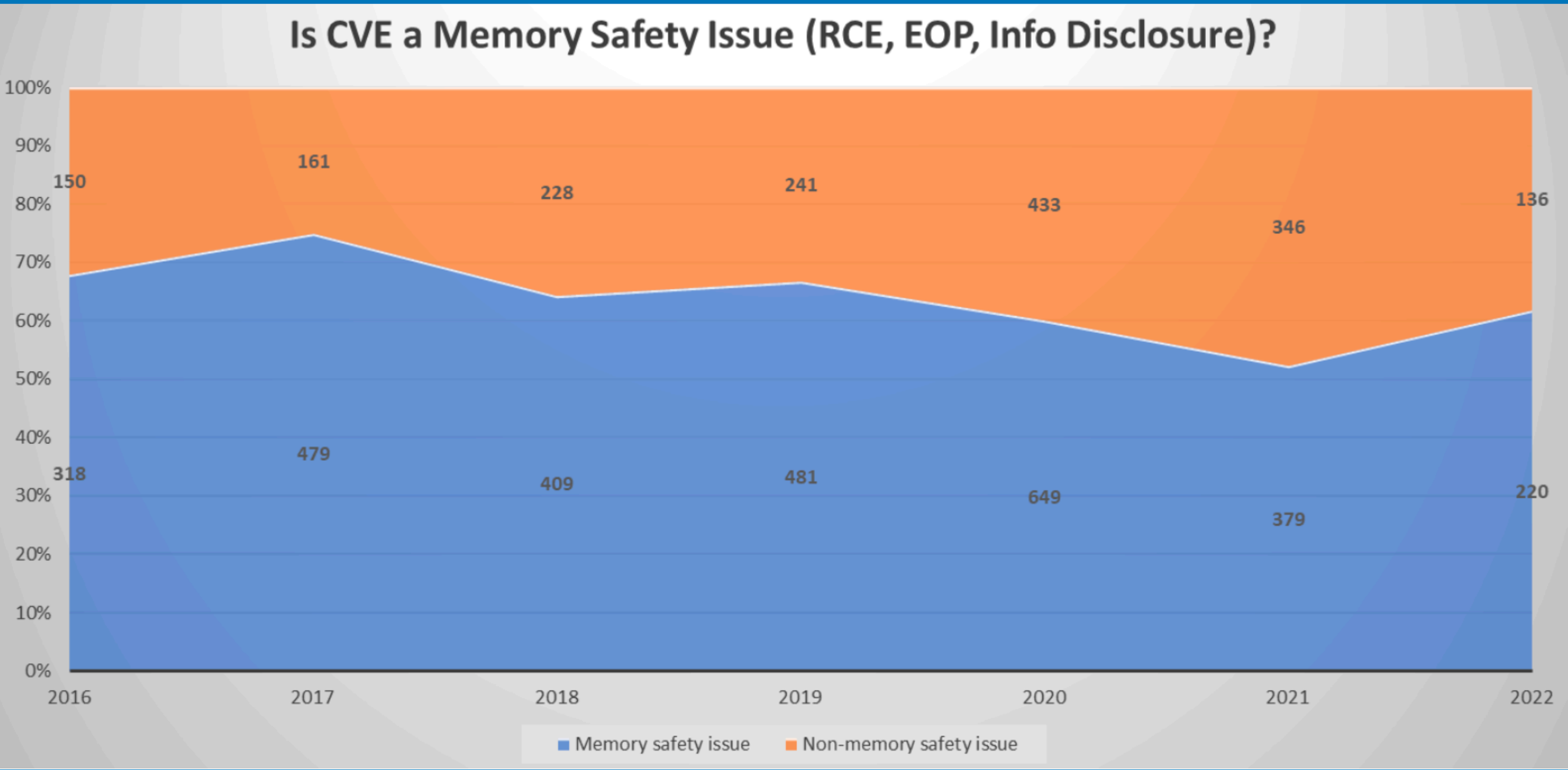
Safe C++?

C++ is inherently unsafe and there's very little* we can do about it

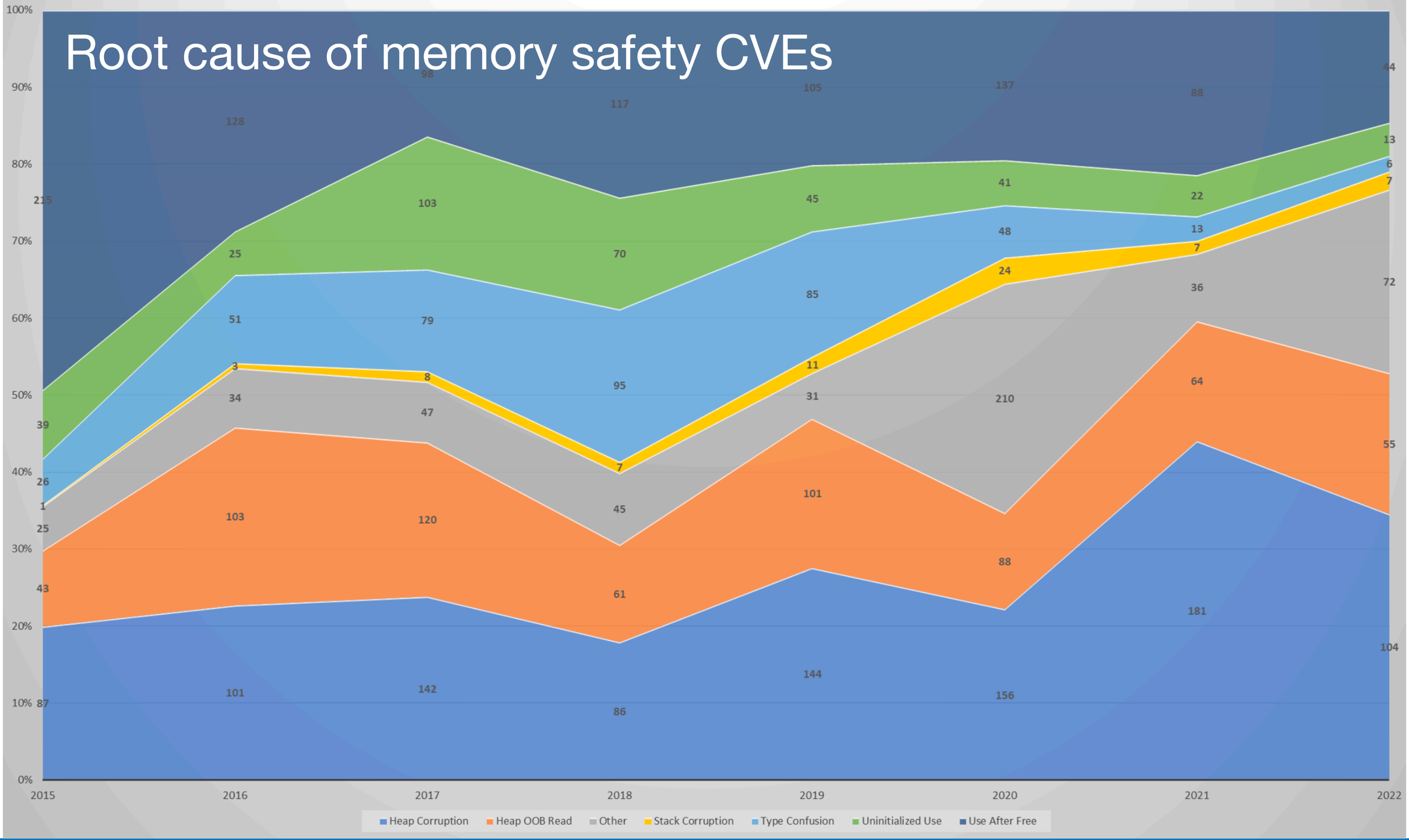
We've known this for years before NSA 😄

media.defense.gov/2022/Nov/10/2003112742/-1/-1/0/CSI_SOFTWARE_MEMORY_SAFETY.PDF

Microsoft CVEs



Root cause of memory safety CVEs



Systems Language Overview

	Rust	C++	C
Object Lifetime	Statically Enforced	Not Enforced, unclear path forward.	No hope
Type Safety	Statically Enforced	Not enforced, unclear path forward.	No hope
Bounds Safety	Enforced at runtime when needed	Could be enforced for STL containers.	No hope
Uninitialized Safety	Statically Enforced	Not enforced, could be enforced w/ breaking change.	Stack could be enforced w/ breaking change.

Bug Classes vs. Mitigations

Vulnerability Class	Deterministic or Probabilistic Mitigation	% of Memory Safety Issue CVE's
Heap Linear Overflow	Deterministic	9.5%
Heap Non-Linear Overflow	Probabilistic	12%
Use-After-Free	Probabilistic	26%
Heap Linear Overread	Deterministic	5.3%*
Heap Non-Linear Overread	Probabilistic	14%*
Uninitialized Memory	Deterministic	12%
Type Confusion	Not Mitigated	14%

We can mostly solve these. Uninitialized doesn't require memory tagging.

We can make progress on these (both detection and exploitability).

CastGuard solves some type confusion, others have no apparent solution.

Microsoft

Ongoing efforts:

- Making a step-changes in our **SDL** operations and making additional investments to meet the evolving needs of cloud and emerging technologies
- Completing our deployment of **CodeQL**, integrated with GitHub Copilot learnings
- Continue to invest in **hardening** C & C++ code
- Standardizing on **Rust** and other memory safe languages (MSLs)
- Contribute 💰 to support the work of the Rust Foundation
- Assist developers making the transition from C, C++, C# to Rust
  - we will continue to invest 💰 in Rust developer tooling

Safety

C++ will never be a 100% safe language

- type safety
- bounds safety
- lifetime safety
- initialization safety
- object access safety
- thread safety
- arithmetic safety

* but it can be much safe(r) with some effort and good tooling 🧰

Hardening C & C++ code

Short term:

- tactical efforts to eliminate attack surface
- block exploit techniques
- statically analyze vulnerabilities
- dynamic analysis & fuzzing
- CLFS signing, heap mitigations, ASLR, CFG, UMFD

Hardening C & C++ code

Long term:

- Combination of software and hardware mitigations to detect & eliminate the most common memory safety issues classes.
- InitAll / Pool Zeroing – Zero initialize stack variables and kernel pool allocations.
- CastGuard– Prevent illegal stack downcasts (type confusion).
- Memory Tagging – Broad impact to a variety of bug classes, hardware feature.

Hardening C & C++ code

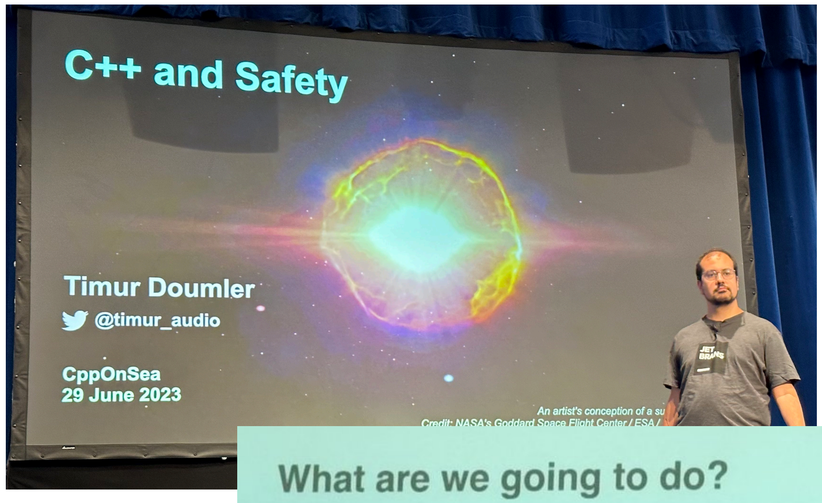
Memory Tagging:

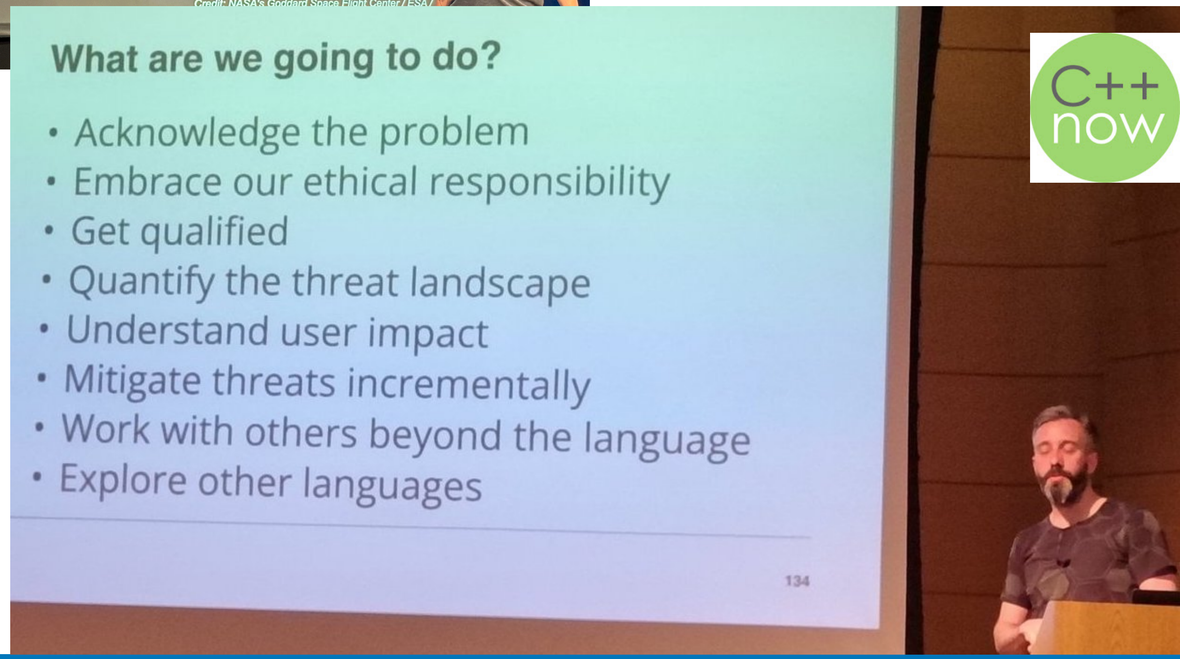
- Helps developers catch bugs (eg. hardware ASAN), stops bugs from being exploitable if they ship to customers.
- Non-trivial CPU and memory overhead, but low enough to enable-by-default in production.
- Google will deploy to Android soon; Apple also expected to deploy.
- Microsoft is actively working w/ silicon partners on Memory Tagging designs that are scalable from small devices to large Azure servers.
- Goal: Enable by default for Windows to make a more reliable and secure OS.
- Goal: Support in Azure, for both Windows and Linux

C++ is under attack...

... and the community is responding 🤷

defense.gov/2022/Nov/CSI_SOFTWARE_MEMORY_SAFETY.PDF





Tradeoffs need to be made...

"To UB, or not to UB"

-- Prince Hamlet

We have not addressed C++ safety until we have eliminated **all** UB.

We can't completely eliminate UB from C++ (for good reasons*).

At minimum, unbounded undefined behaviors (that represent a single point of failure) should be eliminated.

➡️ C++ will never be a safe language (guarantees)

Choices


perf compat ABI
expressive correct
portable complex
safe


C++

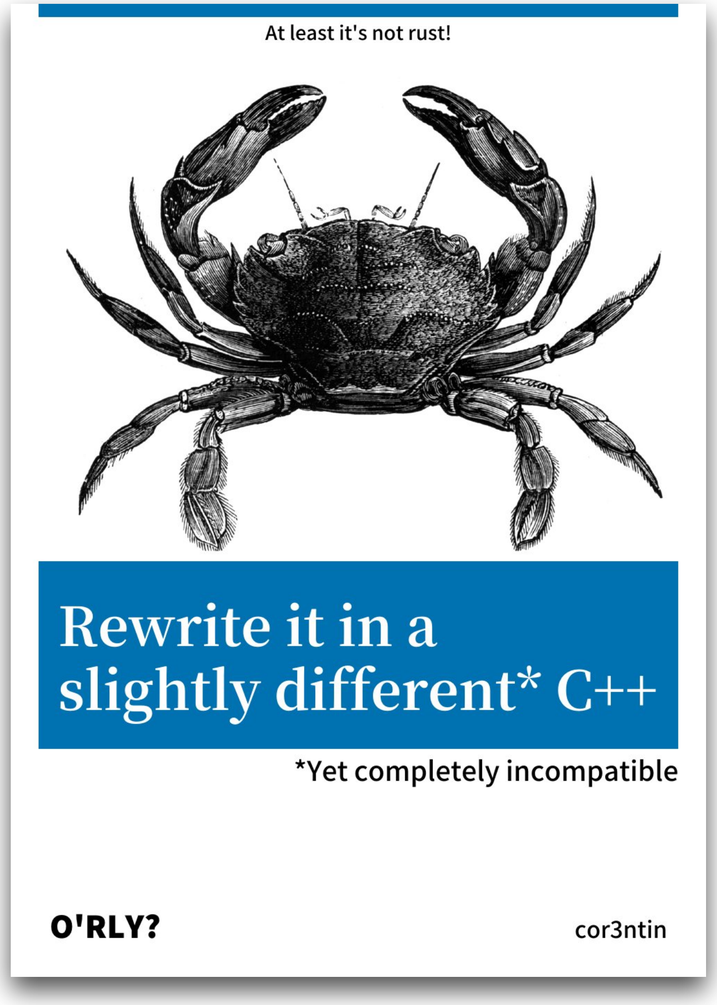
Safety

An excellent essay on the subject of safety: *"If we must, let's talk about safety"*

cor3ntin.github.io/posts/safety/

-- Corentin Jabot

- A cakewalk and eating it too
- Borrowing the borrow checker
- But we care about safety, right?
- Dogma
- Down with Safety!
- UB
- Correct by confusion
- ++(C++) / Rust



Lifetime Safety

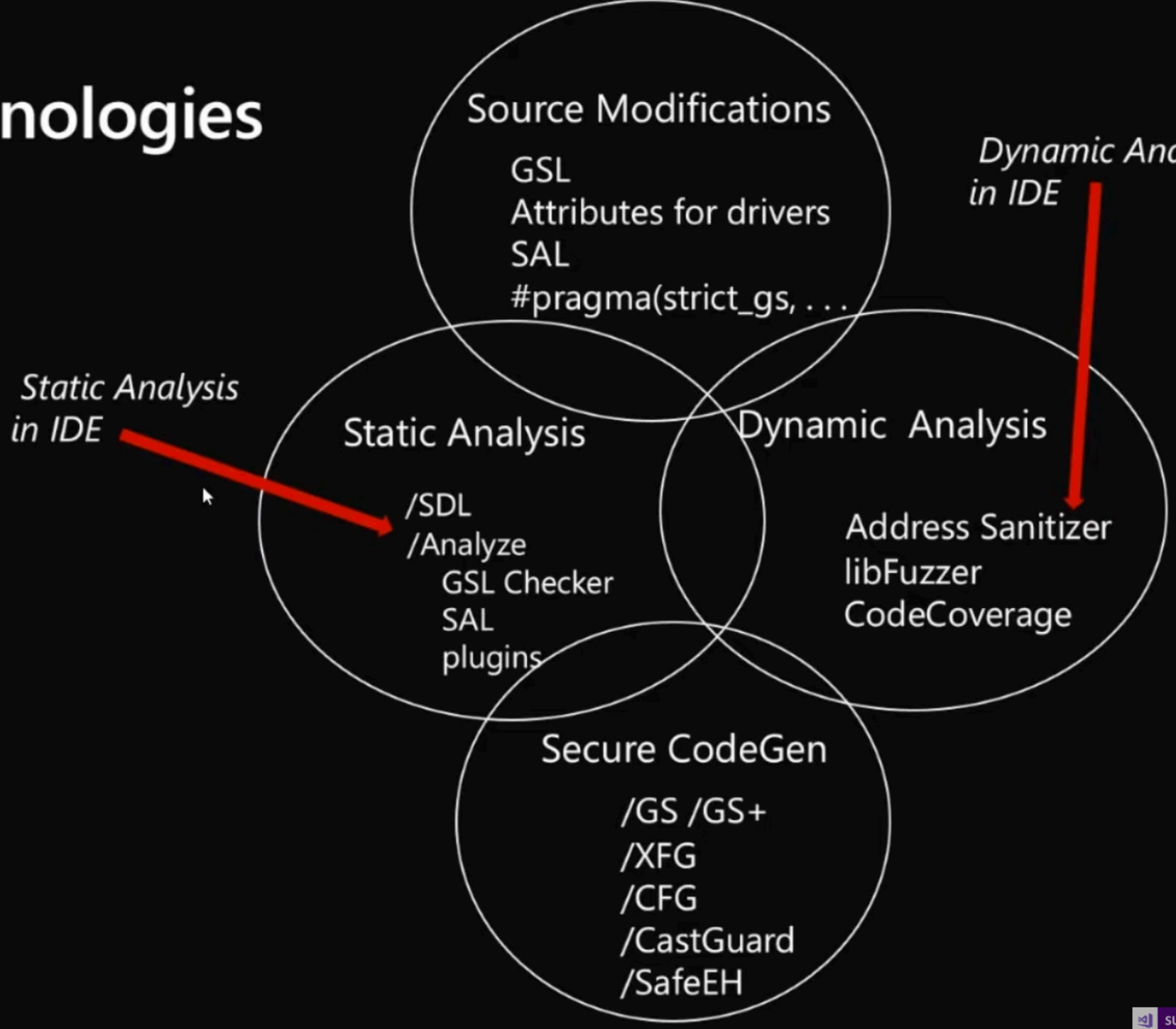
- garbage collector 😱
- dynamic memory analysis (ASan)
- statically enforce rules on references:
- multiple immutable refs II unique mutable ref
    - by compiler/language:
    - borrow checker (Rust, Circle*)
        - mutable value semantics (Val Hylo)
        - no direct mutation (Haskell & other pure functional languages)
        - by tooling (static lifetime analysis):
    - clang-tidy
        - MSVC
        - other commercial analyzers (plenty of them)
        -

The new C++ "AAA"

~~AAA (almost always auto)~~

AAA (almost always analyze)

C++ Security Technologies



youtube.com/watch?v=i8_RfDAEjMs

[[lifetimebound]]

Static Analysis `lifetime annotations for C++`

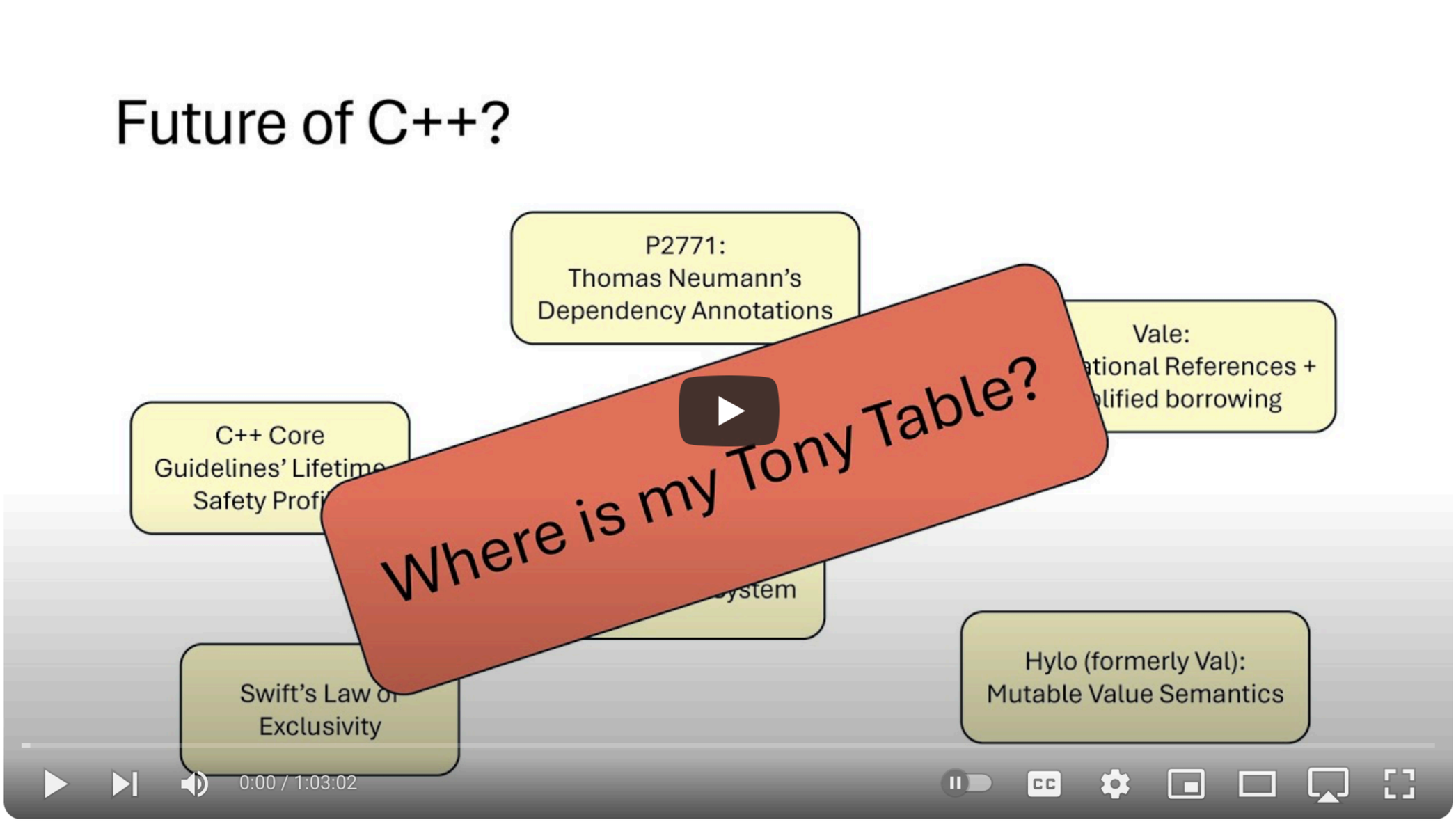
~NEW:

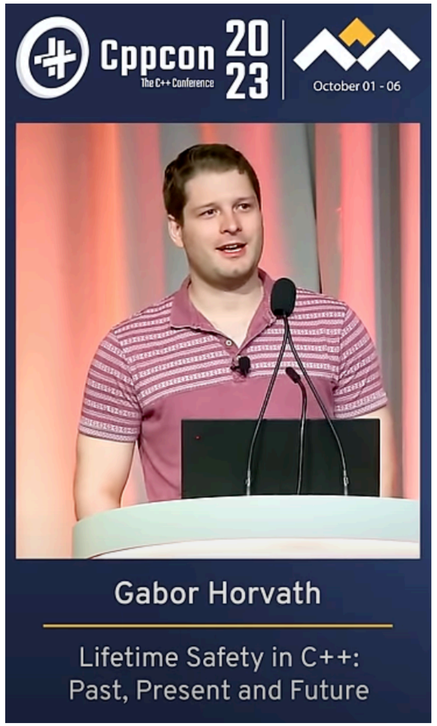
`[[clang::lifetimebound]]` and `[[msvc::lifetimebound]]`

discourse.llvm.org/t/rfc-lifetime-annotations-for-c/61377

learn.microsoft.com/en-us/cpp/code-quality/C26815

youtube.com/watch?v=fe6yu9AQIE4





Lifetime Safety in C++: Past, Present and Future - Gabor Horvath - CppCon 2023

youtube.com/watch?v=PTdy65m_gRE

safe(r)

C++ will never be a safe language*

* but it can be much safe(r) with some effort and good tooling 🧰

RiiR

Just rewrite it in **Rust** 🦀

Microsoft Azure security evolution: Embrace secure multitenancy, Confidential Compute, and Rust

By Jeffrey Cooperstein Partner Software Architect, Azure Security

microsoft-azure-security-evolution-embrace-secure-multitenancy-confidential-compute-and-rust/

Microsoft Azure security evolution: Embrace secure multitenancy, Confidential Compute, and Rust

By Jeffrey Cooperstein Partner Software Architect, Azure Security

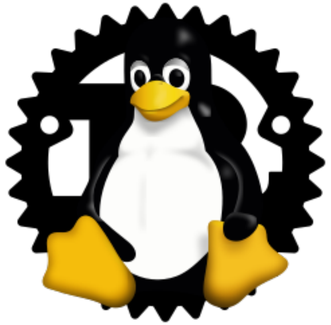
🔒 **trust compute**

microsoft-azure-security-evolution-embrace-secure-multitenancy-confidential-compute-and-rust/

Really?

Rust in the Linux `kernel` (since 6.1)

-- with Linus Torvalds' blessing



The first Rust modules start to make their way into the Linux kernel (6.3+)

Ubuntu has done all the work to provide the right toolchain in the distro and custom kernel patches (SAUCE) that allow easier acquisition and build of Rust modules.

ubuntu

wikipedia.org/wiki/Rust_for_Linux

* but not getting all the love 🤷

Really?

Rust already in the Windows 11 `kernel` (May 2023)

```
C:\Windows\System32>dir win32k*
 Volume in drive C has no label.
 Volume Serial Number is E60B-9A9E

 Directory of C:\Windows\System32

04/15/2023  09:50 PM           708,608 win32k.sys
04/15/2023  09:49 PM         3,424,256 win32kbase.sys
04/15/2023  09:49 PM           110,592 win32kbase_rs.sys
04/15/2023  09:50 PM         4,194,304 win32kfull.sys
04/15/2023  09:49 PM            40,960 win32kfull_rs.sys
04/15/2023  09:49 PM            69,632 win32kns.sys
04/15/2023  09:49 PM            98,304 win32ksgd.sys
               7 File(s)      8,646,656 bytes
               0 Dir(s)  116,366,049,280 bytes free
```

_rs = Rust!

Rusty Windows

So this happened 👀 (public announcement, April 2023)

Ported Windows 11 core components from C++ to Rust

- `DirectWrite`
- `GDI`
- `...` 🤫

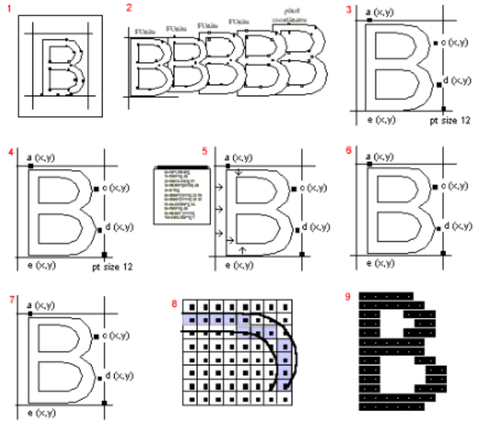
youtube.com/watch?v=8T6ClX-y2AE&t=2703s

Rust in Windows

Learn by doing: **Exploration → Flighting → Production** (crawl → walk → run)

- Direct impact: Improve security
- Gain experience with transitioning to Rust in production
- Costs of learning Rust?
- Costs of porting to Rust?
- Costs of writing new Rust components?
- Is the full pipeline of Rust tooling ready?
- Debugging, perf, cross-platform, POGO, etc.
- Costs of maintaining a hybrid **C++/Rust** codebase?

DWrite - DWriteCore



DWrite is a full stack for text analysis, layout & rendering:

- DWrite ships in Windows (dwrite.dll)
- DWriteCore is cross-platform: Windows, Linux, Android, iOS, macOS
- Office depends on DWrite(Core)
- Rust port work began in 2020. DWriteCore is now ~152 KLOC of Rust
- DWriteCore *internally* uses COM-like interfaces:
  - these were a good integration point for C++/Rust, and provided natural boundaries for incremental porting
- DWriteCore *public* APIs are all COM
  - Rust code is directly callable from app code, through COM interfaces

DWrite Internals

Layout (10 KLOC)

- Line layout, justification
- Text run management: bold, italics, font face, underline, etc.
- Font fallback: Most fonts don't contain all glyphs (e.g. emoji)

Shaping (36 KLOC) + OTLS (18 KLOC)

- Complex script-specific layout: Thai, Indic, Arabic, Hebrew, Hangul, etc.
- Mandatory for complex scripts
- Many are driven by hand-written FSMs
- Complex transformation rules stored in font files (OpenType)
- Transforms sequences of glyphs, e.g. ligatures, connected scripts

Total ported code ~= 152 KLOC (some modules not shown). (Precise counts are complicated, due to test code.)

All code is 100% safe code, except at C++ boundary

Not all parts of DWrite are shown; just those relevant to port

Unicode Analysis (6 KLOC)

- Very large property tables
- Defined by Unicode standard

Glyph Data + Glyph Rendering (24 KLOC)

- Computes vector curves, runs bytecode programs (!!) from font files to adjust them
- Rasterizes vector curves to bitmaps
- Provides metrics (advance width, x-height, side bearings)
- Scales bitmaps for high-density scripts (e.g. Chinese)

Porting Time

- TrueType
  - ~ 2 months (1 dev, experienced in Rust) for the core functionality
  - ~ 2 months for exhaustive comparison testing and regression fixing
- Shaping + OTLS
  - ~ 2 months
  - ~ 1 month for comparison testing and regression fixing
  - ~ 2 weeks for performance improvements
- Layout
  - ~ 1.5 months
  - ~ 2 weeks for testing / regression fixing
- Unicode analysis
  - ~ 2 weeks
  - Low rate of regressions; very data-oriented

Win32k GDI - Rust

Ported the REGION components:

- Models overlapping controls (e.g., windows) in GDI
- "Leaf node" data type: few dependencies, many dependents
- Old (late 80s, early 90) and perf critical (designed for i286/i386)
- Maintenance nightmare: open-coded vector resizing and ref-counting
- Windows boots with the Rust version, and all GDI tests pass
- In flight testing, to prove viability

Win32k GDI - Rust

- Performance of ported code is excellent
  - Office tests, micro-benchmarks
- This work has driven contributions to upstream Rust project
- Lots of calls to extern C/C++ functions => still a lot of unsafe code
- Unsafe area is reducing as we port more and more code to Rust
- Milestone: able to write a SysCall in completely safe code

Oxidation

More oxidation 🦀 efforts in progress...

Rust Fact vs. Fiction

5 Insights from Google's Rust journey

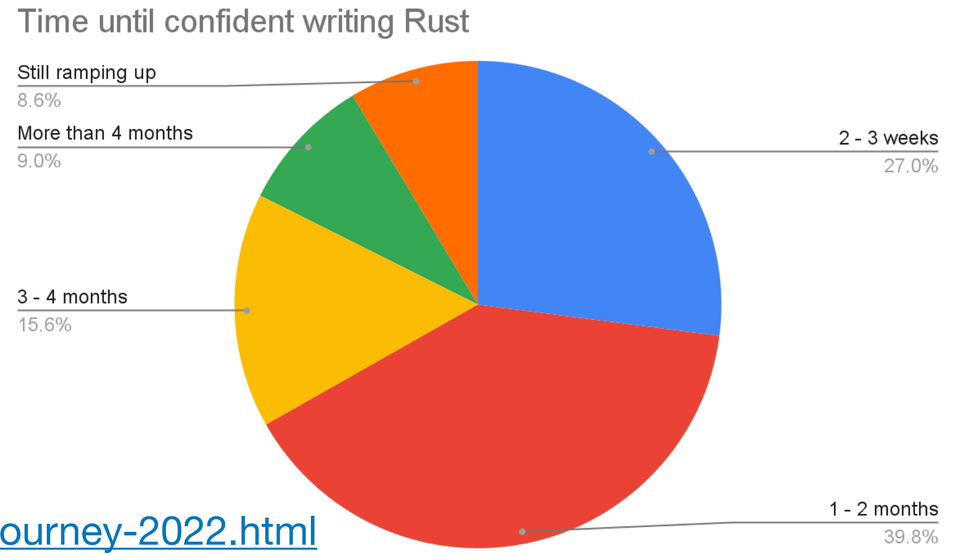
Rumor 1: Rust takes more than 6 months to learn – Debunked

Rumor 2: The Rust compiler is not as fast as people would like – Confirmed

Rumor 3: Unsafe code and interop are always the biggest challenges – Debunked

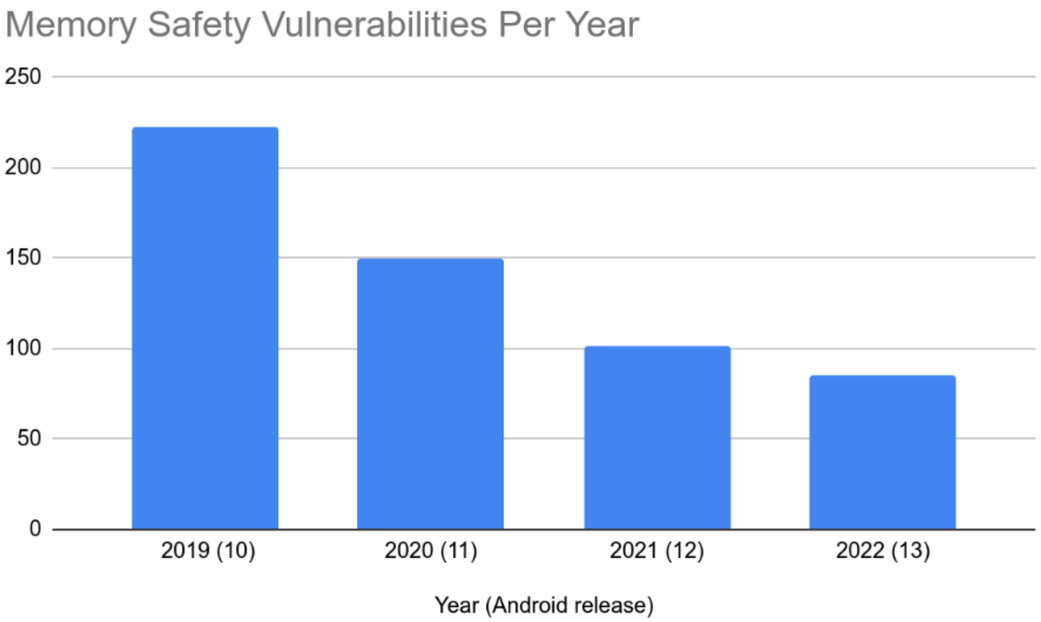
Rumor 4: Rust has amazing compiler error messages – Confirmed

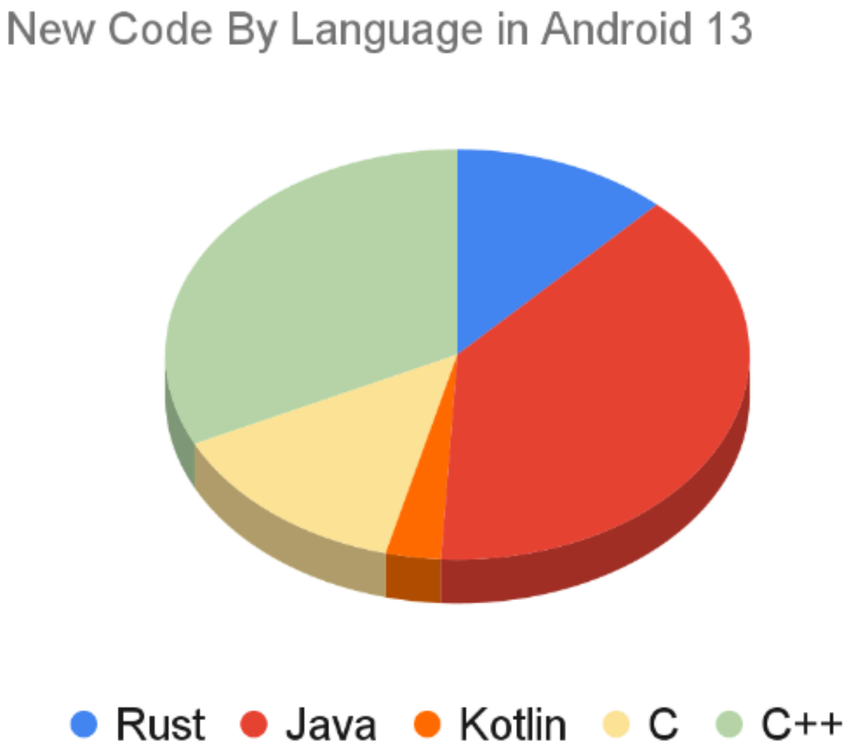
Rumor 5: Rust code is high quality – Confirmed



opensource.googleblog.com/2023/06/rust-fact-vs-fiction-5-insights-from-googles-rust-journey-2022.html

security.googleblog.com/2023/10/bare-metal-rust-in-android





security.googleblog.com/2022/12/memory-safe-languages-in-android-13

C++ Interop

“Based on historical vulnerability density statistics, Rust has proactively prevented hundreds of vulnerabilities from impacting the Android ecosystem. This investment aims to expand the adoption of Rust across various components of the platform.”

– Dave Kleidermacher, Google Vice President of Engineering, Android Security & Privacy

“While Rust may not be suitable for all product applications, prioritizing seamless interoperability with C++ will accelerate wider community adoption, thereby aligning with the industry goals of improving memory safety.”

– Royal Hansen, Google Vice President of Safety & Security

foundation.rust-lang.org/news/google-contributes-1m-to-rust-foundation-to-support-c-rust-interop-initiative/

Rust and C++ interoperability

It’s important for Rust to be able to call C++ functions in a way that meets the following criteria:

- No need for excessive unsafe keyword
- No overhead in the general case
- No boilerplate or re-declarations / No C++ annotations
- Broad type support - with safety
- *Ergonomics* - with safety

🚧 There's progress in Rust community in solving some of these problems.

⚙️ cxx, autocxx, bindgen, cbindgen, diplomat, crubit

Learning Rust

First Encounter

Lars Marowsky-Brée 😷
@larsmb@mastodon.online

@lina @mstrohm There's two reactions to someone who knows C(++) to deal with learning about Rust -
Either you have a joyful breakdown because someone understands your PTSD and offers you a safe haven,
Or you end up defending your PTSD because you've invested so much into the past.

Aug 31, 2024, 07:33 PM · · Web

mastodon.online/@larsmb/113057830402545219

Learning

Google's **Comprehensive Rust** 🦀 Training

google.github.io/comprehensive-rust/

- Rust Fundamentals: 4 days
- + Android
  - includes interoperability with C, C++, and Java
- + Chromium
  - includes interoperability with C++ and how to include third-party crates in Chromium
- + Bare-metal: kernel & embedded development
- + Concurrency
  - classical concurrency (preemptively scheduling using threads and mutexes)
  - async/await concurrency (cooperative multitasking using futures)

Day 1: Morning

3. Welcome

4. Hello, World

5. Types and Values

6. Control Flow Basics

Day 1: Afternoon

7. Welcome

8. Tuples and Arrays

9. References

10. User-Defined Types

Day 2: Morning

11. Welcome

12. Pattern Matching

13. Methods and Traits

Day 2: Afternoon

14. Welcome

15. Generics

16. Standard Library Types

17. Standard Library Traits

Day 3: Morning

18. Welcome

19. Memory Management

20. Smart Pointers

Day 3: Afternoon

21. Welcome

22. Borrowing

23. Lifetimes

Day 4: Morning

24. Welcome

25. Iterators

26. Modules

27. Testing

Day 4: Afternoon

28. Welcome

29. Error Handling

30. Unsafe Rust

Crates Audit

`cargo vet` ➡️ `audits.toml`

Tool to help projects ensure that third-party Rust dependencies have been audited by a trusted entity.

Downsides:

- review format is too sparse
- just marking a crate version as safe-to-run or safe-to-deploy
- add optional notes (but no more details)
- no ability to track internal crate usage, relative to reviews

mozilla.github.io/cargo-vet/

googleblog.com/2023/05/open-sourcing-our-rust-crate-audits

Rust Crate Review System

A system that records guidance from Microsoft developers on using Rust crates, both public and internal.

Questions this system helps answer:

- What Rust crates should my Microsoft product use, or not use?
- How should I evaluate public Rust crates I'm considering using, and record the evaluation?
- What are the preferred crates for particular purposes?

Rust Crate Review System

Rust Crate Review System

- External crate dependencies, come with the inherent risk in the form of potential security issues, stability issues, and support and maintenance related issues.

Rust Crate Review System

- External crate dependencies, come with the inherent risk in the form of potential security issues, stability issues, and support and maintenance related issues.

Rust Crate Review System

- External crate dependencies, come with the inherent risk in the form of potential security issues, stability issues, and support and maintenance related issues.
- Proactively ensure that MSFT Rust ecosystem is built on the stable and thriving part of the OSS Rust ecosystem, lowering the risk of being affected eg. by a vulnerability reported on a pet-project crate with a single owner who is not maintaining it anymore.

Rust Crate Review System

- External crate dependencies, come with the inherent risk in the form of potential security issues, stability issues, and support and maintenance related issues.
- Proactively ensure that MSFT Rust ecosystem is built on the stable and thriving part of the OSS Rust ecosystem, lowering the risk of being affected eg. by a vulnerability reported on a pet-project crate with a single owner who is not maintaining it anymore.

Rust Crate Review System

- External crate dependencies, come with the inherent risk in the form of potential security issues, stability issues, and support and maintenance related issues.
- Proactively ensure that MSFT Rust ecosystem is built on the stable and thriving part of the OSS Rust ecosystem, lowering the risk of being affected eg. by a vulnerability reported on a pet-project crate with a single owner who is not maintaining it anymore.
- A set of unbiased Rust crate evaluation criteria, used for assessment of adoptability of third-party crates by any internal Rust project, lowering the company's vulnerability naturally introduced by depending on third-party OSS solutions.

Rust Crate Review System

- External crate dependencies, come with the inherent risk in the form of potential security issues, stability issues, and support and maintenance related issues.
- Proactively ensure that MSFT Rust ecosystem is built on the stable and thriving part of the OSS Rust ecosystem, lowering the risk of being affected eg. by a vulnerability reported on a pet-project crate with a single owner who is not maintaining it anymore.
- A set of unbiased Rust crate evaluation criteria, used for assessment of adoptability of third-party crates by any internal Rust project, lowering the company's vulnerability naturally introduced by depending on third-party OSS solutions.

Rust Crate Review System

- External crate dependencies, come with the inherent risk in the form of potential security issues, stability issues, and support and maintenance related issues.
- Proactively ensure that MSFT Rust ecosystem is built on the stable and thriving part of the OSS Rust ecosystem, lowering the risk of being affected eg. by a vulnerability reported on a pet-project crate with a single owner who is not maintaining it anymore.
- A set of unbiased Rust crate evaluation criteria, used for assessment of adoptability of third-party crates by any internal Rust project, lowering the company's vulnerability naturally introduced by depending on third-party OSS solutions.
- A unified, unbiased, highly automatable crate scoring system used in Microsoft.


ONE DOES NOT SIMPLY
REWRITE IN RUST
makeameme.org

Unleashing 🦀 The Ferris Within

NDC { TechTown }

September 2024

🐦 @ciura_victor
🦣 @ciura_victor@hachyderm.io
🦋 @ciuravictor.bsky.social

Victor Ciura
Principal Engineer
Rust Tooling @ Microsoft