Preventing Rust Programs from Becoming C Programs: Introduction

Anne Mulhern

Red Hat, Inc.

May 19, 2022

Audience

Who should want to attend this talk?

- People who have a general interest in the Rust language.
- People who have experience with the Rust language and may be interested in Rust language analysis tools (other than those already provided by the Rust toolchain itself).
- People who write unsafe code in Rust and wonder if they can get some help with writing it correctly.
- People who have experience with programming in Rust and have encountered an error that can be traced to unsafety.

Me

Who is speaking right now?

- I go by "mulhern", not by my legal first name.
- I'm the Stratis[1] team tech-lead.
- I am not the principal researcher for this project; I proposed it.
- I have a Ph.D. in Computer Sciences.

[1] https://stratis-storage.github.io/

Expectations

What this talk will be like.

- You are encouraged to be forthcoming with your questions.
- I'll provide some background, then I'll yield the floor[2] to the principal researchers.

[2] metaphorically speaking, of course

My Goal

Mostly, I was interested in demonstrating how great Rust is in spite of...

- Rust is quite hard to learn.
- Writing programs in Rust is fairly hard.
- Refactoring large Rust programs can be arduous.
- etc.

Why Rust?

What makes choosing to develop a project in Rust worth it?

- So many things I can't even begin to discuss them all.
- One way to begin is to compare the experience of developing in Rust with the experience of developing in other languages: developing in Rust is more fun than developing in other languages because virtually all "boring" bugs are eliminated.

Why not Python instead of Rust?

What are the “boring” bugs characteristic of Python?

Type-error induced stack trace

```
Traceback (most recent call last):
  File "_main.py", line 43, in the_func
    result.func(result)
  File "/_parser.py", line 89, in wrapped_func
    check_version()
  File "_version.py", line 39, in check_version
    version = Manager0.Pet(get_object(TOP_OBJECT))
AttributeError: type object 'Version' has no ...
```

Why not C instead of Rust?

What are the “boring” bugs characteristic of C?

Memory errors

```
Segmentation fault (Core dumped)
```

These don't happen in Rust

So much less tedium

- Rust has:
  - a flexible and expressive type-system with static checking (like O'Caml, Haskell, etc.)
  - a borrow checker (no other production language that I know has this)
- *Developing software projects in Rust is more fun*

We ended up focusing on C for comparison

Rust has a special relationship with C

- It is a “systems language”, it is frequently used in the same domain.
- Rust “includes” C.

Rust = Rust + C

Rust has `unsafe` blocks.

A typical Rust function with unsafety

```
fn my_func(v: &'a Value) -> Result<(), Error> {
    let ptrs = unsafe {
        slice::from_raw_parts(v.things,
                              v.count as usize)
    };
    let mut stuff = Vec::new();
    for ptr in ptrs {
        stuff.push(unsafe { ... ptr ... });
    }
    ...
}
```

Why use unsafe?

Why get C in your Rust code?

- Wrapping C libraries
- Optimizing data structures
- etc.

Rules for `unsafe`

What is unsafe code and how is unsafety propagated?

- Base case:
  - operations that are by definition unsafe (the C-like operations)
  - declared unsafe:
    - ★ any `unsafe` function
    - ★ any `unsafe` block
- Inductive case: If a function invokes an unsafe function then:
  - the invoking function must also be marked as unsafe OR
  - the invocation must be within an `unsafe` block

Goals with `unsafe`

What should you, as programmer, aim for, when writing a library that includes unsafety?

- Minimize the static size of the `unsafe` code, to make reasoning about it easier.
- Ensure that code that is not labelled unsafe meets the expectations that programmers have of pure Rust.

Pitfalls with `unsafe`

Can you, as a programmer, err, so that programming in Rust is no better than programming in C?

- You can allow the unsafety to escape and be present in your functions that are not labeled `unsafe`

Are real programs vulnerable?

Am I making a big deal out of nothing?

- There are 5900 occurrences of the `unsafe` keyword in the source of the code that `stratisd`[3] depends on.
- There is a database of Rust security vulnerabilities[4], a signficant proportion are the kinds of unsafety, i.e., boring bugs that I've just discussed.

[3] `The principle application of the Stratis project.`

[4] `https://rustsec.org/advisories/`

Thanks

Next up

- The principle researchers
- Any last questions for me before I hand over?