Building Safe and Secure Systems in Rust

Mingshen Sun | Baidu X-Lab, USA

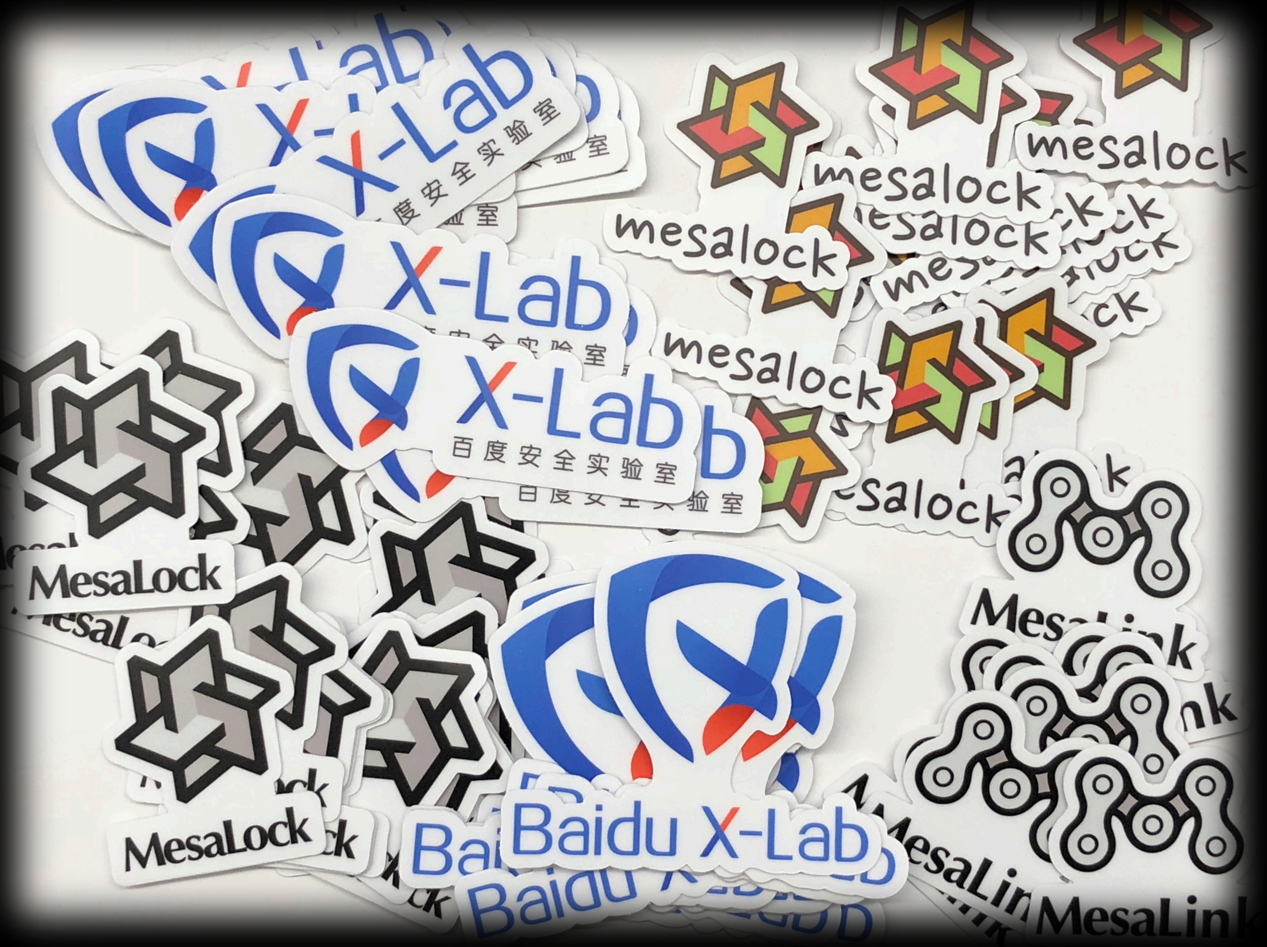
December 2018

RustRush, Moscow

About Me

- Senior Security Researcher in **Baidu X-Lab**, USA
- System security, mobile security, IoT security, and car hacking
- Maintaining open-source projects: MesaLock Linux, MesaPy, TaintART, Pass for iOS, etc.
- `mssun @ GitHub | https://mssun.me`

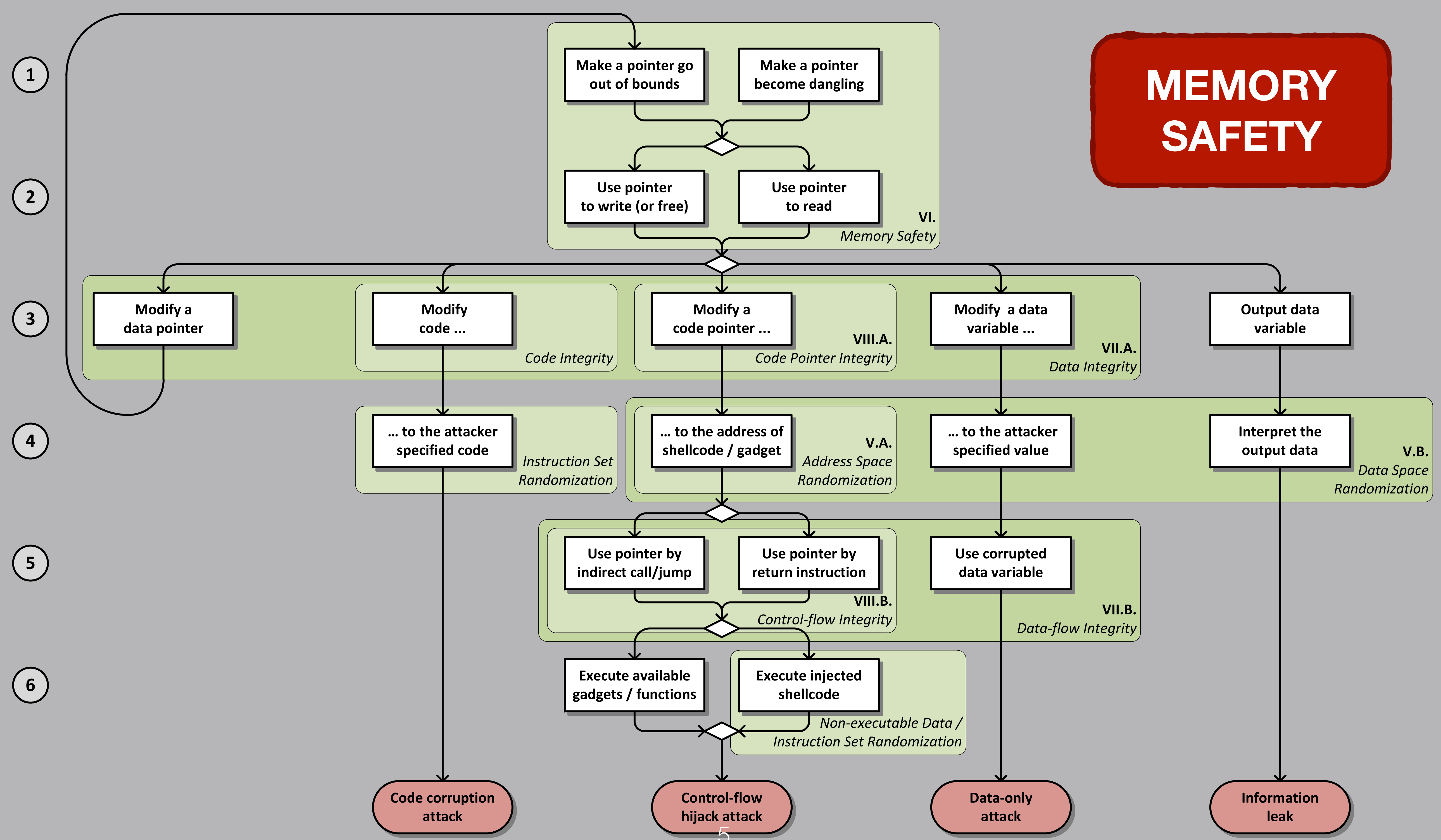
Baidu X-Lab



Outline

- Building **safe** and **secure** systems in Rust
- **Challenges**, **lessons** learned, and open **questions**

SoK: Eternal War in Memory
Laszlo Szekeres, Mathias Payer, Tao Wei, Dawn Song
Proceedings of the 2013 IEEE Symposium on Security and Privacy



1
2
3
4
5
6
Make a pointer go out of bounds
Make a pointer become dangling
Use pointer to write (or free)
Use pointer to read
VI.
Memory Safety
MEMORY SAFETY
Modify a data pointer
Modify code ...
Code Integrity
Modify a code pointer ...
VIII.A.
Code Pointer Integrity
Modify a data variable ...
VII.A.
Data Integrity
Output data variable
... to the attacker specified code
Instruction Set Randomization
... to the address of shellcode / gadget
V.A.
Address Space Randomization
... to the attacker specified value
Interpret the output data
V.B.
Data Space Randomization
Use pointer by indirect call/jump
Use pointer by return instruction
VIII.B.
Control-flow Integrity
Use corrupted data variable
VII.B.
Data-flow Integrity
Execute available gadgets / functions
Execute injected shellcode
Non-executable Data / Instruction Set Randomization
Code corruption attack
Control-flow hijack attack
Data-only attack
Information leak

Approaches to Mitigate Memory Corruption Errors

- Program analysis like symbolic execution: **KLEE**
- Memory-checking virtual machine: **Valgrind**
- Compiler instrumentation: **AddressSanitizer**
- Fuzzing: **AFL, libFuzzer**
- Formal verification: **Seahorn, Smack, Trust-in-Soft**

Approaches to Mitigate Memory Corruption Errors

- ~~Program analysis like symbolic execution: **KLEE**~~
- ~~Memory-checking virtual machine: **Valgrind**~~
- ~~Compiler instrumentation: **AddressSanitizer**~~
- ~~Fuzzing: **AFL, libFuzzer**~~
- ~~Formal verification: **Seahorn, Smack, Trust-in-Soft**~~
- **"Safe" programming languages: Rust, Go, etc**

Building Safe and Secure Systems in Rust

- **Safe**: safe memory access, safe concurrency
- **Secure**: less vulnerabilities, reduced attack surfaces

Building Safe and Secure Systems in Rust

- **operating system**: TockOS, RedoxOS
- **compiler**: Rust
- **network service**: DNS, TLS, web server, etc.
- **database**
- **browser**: Servo, CSS engine, etc.

Baidu X-Lab ❤️ Rust

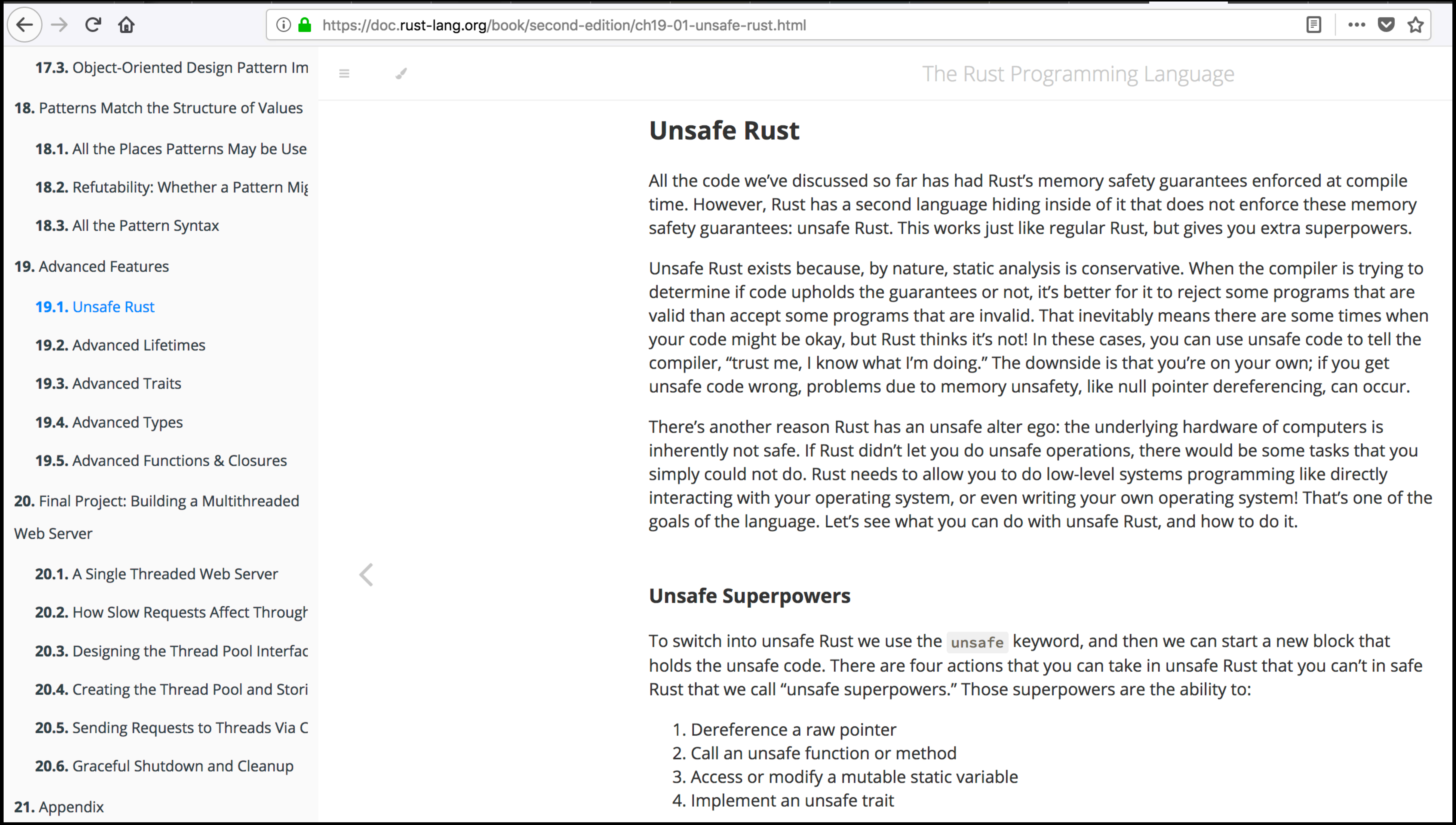
- **MesaLock Linux**: a memory-safe Linux distribution
- **MesaBox**: a collection of core system utilities written in Rust
- **MesaLink**: a memory-safe and OpenSSL-compatible TLS library
- **MesaPy**: secure and fast Python based on PyPy
- **Rust SGX SDK**: provides the ability to write Intel SGX applications in Rust
- and many more ...

Challenges, Lessons Learned, and Open Questions

Challenges

- Rust language and ecosystem
- Unsafe Rust
- Foreign Function Interface (FFI)
- Challenges in hybrid memory model

Memory safe? Meh...



https://doc.rust-lang.org/book/second-edition/ch19-01-unsafe-rust.html
The Rust Programming Language

Unsafe Rust

All the code we've discussed so far has had Rust's memory safety guarantees enforced at compile time. However, Rust has a second language hiding inside of it that does not enforce these memory safety guarantees: unsafe Rust. This works just like regular Rust, but gives you extra superpowers.

Unsafe Rust exists because, by nature, static analysis is conservative. When the compiler is trying to determine if code upholds the guarantees or not, it's better for it to reject some programs that are valid than accept some programs that are invalid. That inevitably means there are some times when your code might be okay, but Rust thinks it's not! In these cases, you can use unsafe code to tell the compiler, "trust me, I know what I'm doing." The downside is that you're on your own; if you get unsafe code wrong, problems due to memory unsafety, like null pointer dereferencing, can occur.

There's another reason Rust has an unsafe alter ego: the underlying hardware of computers is inherently not safe. If Rust didn't let you do unsafe operations, there would be some tasks that you simply could not do. Rust needs to allow you to do low-level systems programming like directly interacting with your operating system, or even writing your own operating system! That's one of the goals of the language. Let's see what you can do with unsafe Rust, and how to do it.

Unsafe Superpowers

To switch into unsafe Rust we use the `unsafe` keyword, and then we can start a new block that holds the unsafe code. There are four actions that you can take in unsafe Rust that you can't in safe Rust that we call "unsafe superpowers." Those superpowers are the ability to:

1. Dereference a raw pointer
2. Call an unsafe function or method
3. Access or modify a mutable static variable
4. Implement an unsafe trait

What is Unsafe Rust?

- All the code we've discussed so far has had Rust's memory safety guarantees enforced at compile time.
- However, Rust has a second language hiding inside of it that **does not enforce** these memory safety guarantees: **unsafe Rust**. This works just like regular Rust, but gives you **extra superpowers**.

Unsafe Superpowers

1. Dereference a **raw** pointer
2. Access or modify a **mutable static variable**
3. Call an unsafe function or method
4. Implement an unsafe trait

Unsafe Superpowers

1. Dereference a raw pointer

Rust

```
unsafe {
    let address = 0x012345usize;
    let r = address as *const i32;
}
```

Read/write arbitrary memory address.

Unsafe Superpowers

2. Access or modify a mutable static variable

Rust

```
static mut COUNTER: u32 = 0;

fn add_to_count(inc: u32) {
    unsafe { COUNTER += inc; }
}

fn main() {
    add_to_count(3);

    unsafe { println!("COUNTER: {}", COUNTER); }
}
```

Data races.

Unsafe Superpowers

3. Call an unsafe function or method

Rust

```
unsafe fn dangerous() {
    let address = 0x012345usize;
    let r = address as *const i32;
}

fn main() {
    unsafe { dangerous(); }
}
```

Call functions may cause undefined behaviors.

Unsafe Superpowers

3. Call an unsafe function or method (external)

Rust

```
extern "C" {
    fn abs(input: i32) -> i32;
}

fn main() {
    unsafe {
        println!("Absolute value of -3 according to C:
{}", abs(-3));
    }
}
```

Call external functions may cause undefined behaviors.

"Unsafe" is agnostic

- **Rust developers**: It's OK. At least you **explicitly** type the **"unsafe" keyword** in the source code, and I know it is "unsafe" before using it.

- **Me**: Wrong. The "unsafe" code could be included in the dependent libraries. Did you review the source code of dependencies?

"Unsafe" is agnostic

Rust

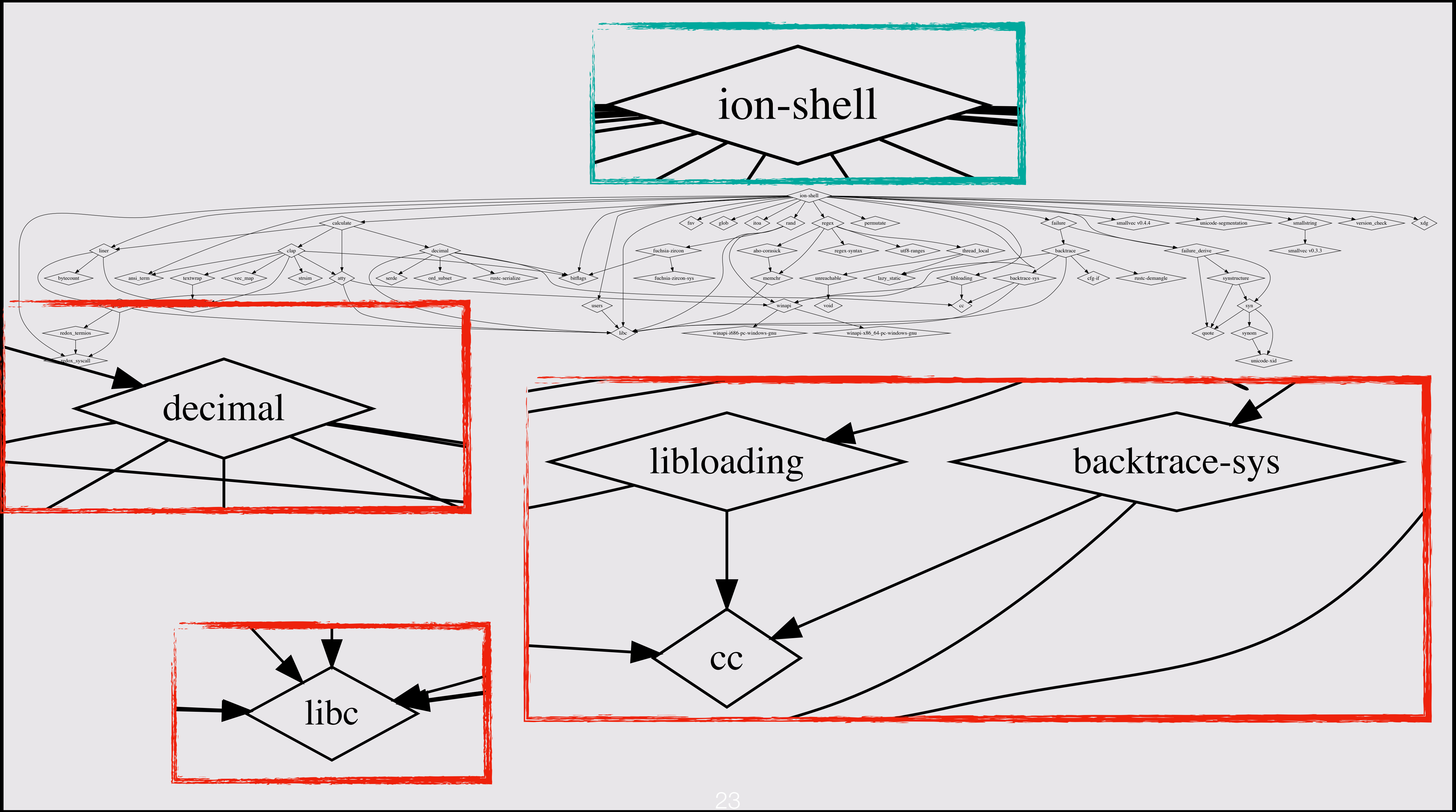
```
Library:

unsafe fn dangerous() {
    let address = 0x012345usize;
    let r = address as *const i32;
}

fn safe_function() {
    unsafe { dangerous(); }
}

Developer:

fn main {
    safe_function();
}
```

some libraries (including the std library) wrap unsafe code and re-export as "safe" functions

Case study: Ion Shell

- Ion is a modern system shell that features a simple, yet powerful, syntax. **It is written entirely in Rust, which greatly increases the overall quality and security of the shell.** It also offers a level of performance that exceeds that of Dash, when taking advantage of Ion's features. While it is developed alongside, and primarily for, RedoxOS, it is a fully capable on other *nix platforms.

Dependency graph of Ion shell



cargo build -vv

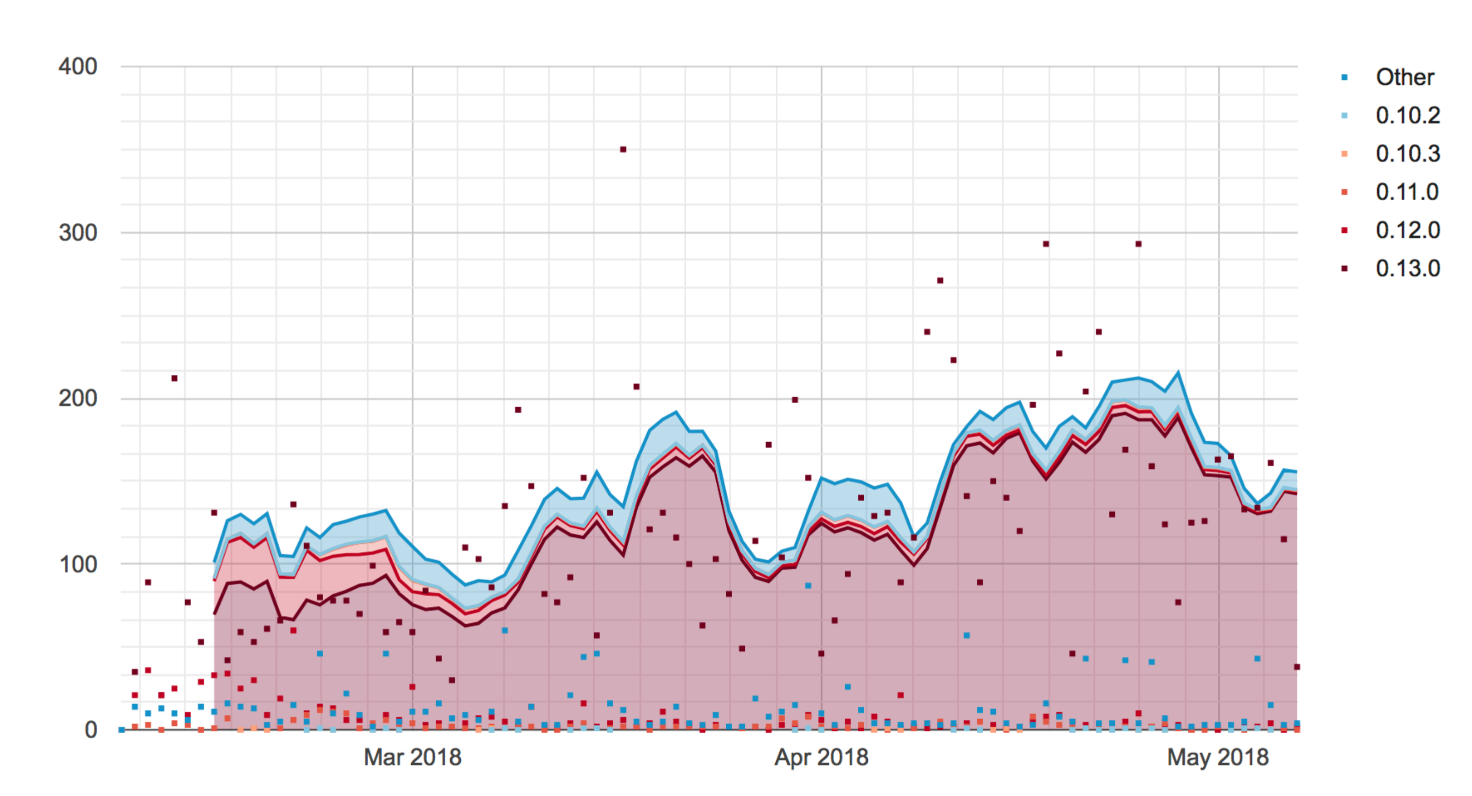
- Build Ion shell again with verbose output.

```
running: "cc" "-O0" "-ffunction-sections" "-fdata-sections"
"-fPIC" "-g" "-m64" "-I" "decNumber" "-Wall" "-Wextra" "-
DDECLITEND=1" "-o" "/Users/mssun/Repos/ion/target/debug/
build/decimal-b8ff0faecf5447ab/out/decNumber/decimal64.o" "-
c" "decNumber/decimal64.c"
```

- decimal crate: Decimal Floating Point arithmetic for rust based on the decNumber library. (http://speleotrove.com/decimal/decnumber.html)
- Ion shell depends on a decimal crate which still uses C code with potential memory safety issues.

Case study: rusqlite

- rusqlite is a Rust library providing SQLite related APIs
- an API wrapper of SQLite written in C
- 38 crates directly depend on rusqlite
- 200 downloads/day



Memory corruption in rusqlite library

- We tried a SQLite type confusion bug (CVE-2017-6991) in rusqlite library
- We can easily trigger the vulnerabilities

Many Birds, One Stone: Exploiting a Single SQLite Vulnerability Across Multiple Software, Siji Feng, Zhi Zhou, Kun Yang, BlackHat USA 17

Rust

```
extern crate rusqlite;
use rusqlite::Connection;

fn main() {
    let conn = Connection::open_in_memory().unwrap();
    match conn.execute("create virtual table a using fts3(b);", &[]) {
        // ...
    }
    match conn.execute("insert into a values(x'4141414141414141');", &[]) {
        // ...
    }
    match conn.query_row("SELECT HEX(a) FROM a", &[], |row| -> String
{ row.get(0) }) {
        // ...
    }
    match conn.query_row("SELECT optimize(b) FROM a", &[], |row| -> String
{ row.get(0) }) {
        // ...
    }
}
```

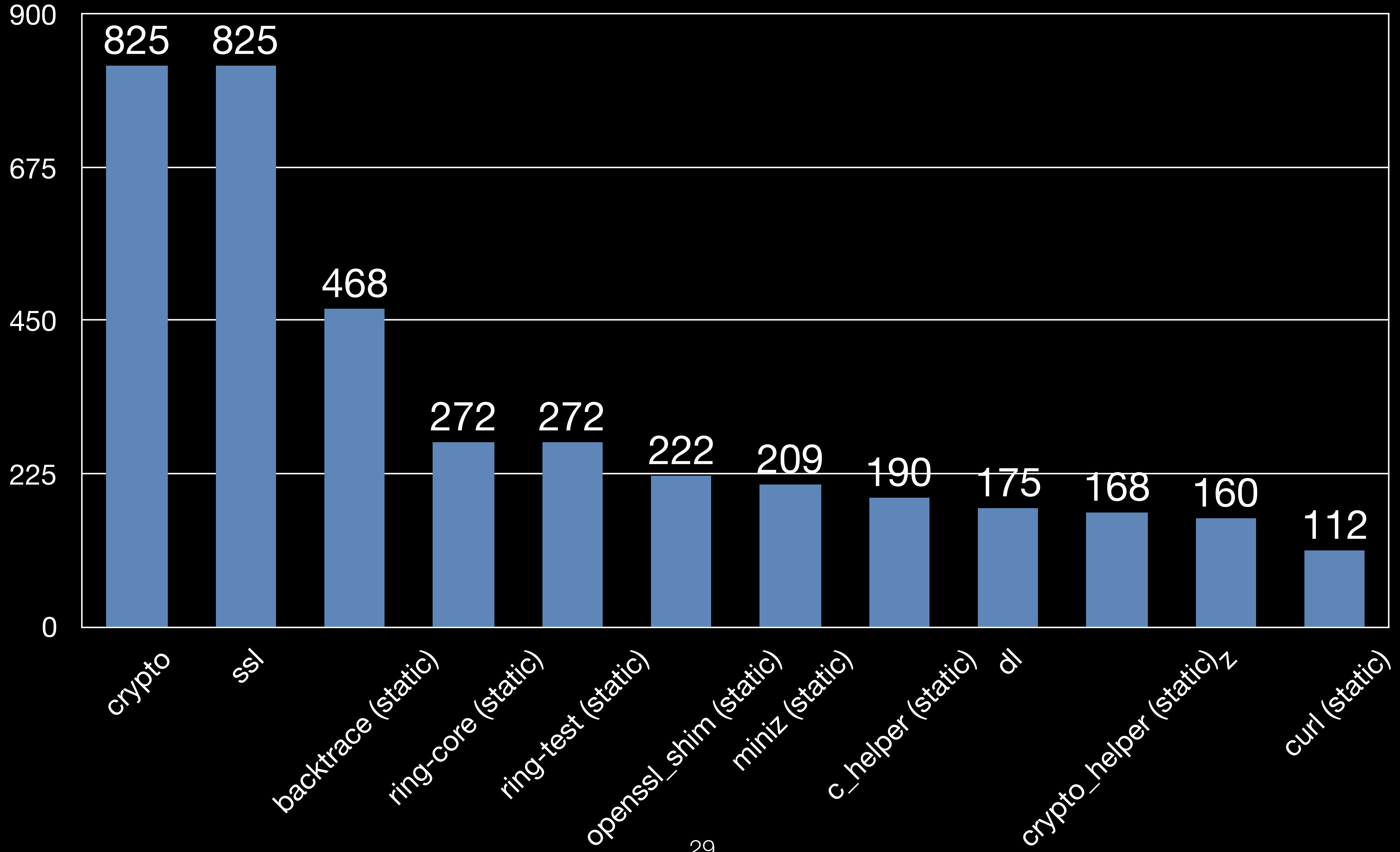
Run

```
$ cargo run
    Finished dev [unoptimized + debuginfo] target(s) in 0.05 secs
     Running `target/debug/rusqlite`
success: 0 rows were updated
success: 1 rows were updated
success: F0634013D87F0000
[1]    31467 segmentation fault  cargo run
```

Data Collection and Study

- **10,693** Rust libraries in crates.io
- **200 million public downloads** in total
- two studies
  - usage of external C/C++ libraries
  - usage of unsafe keywords

Usage of external libraries (>= 100)



“unsafe” code

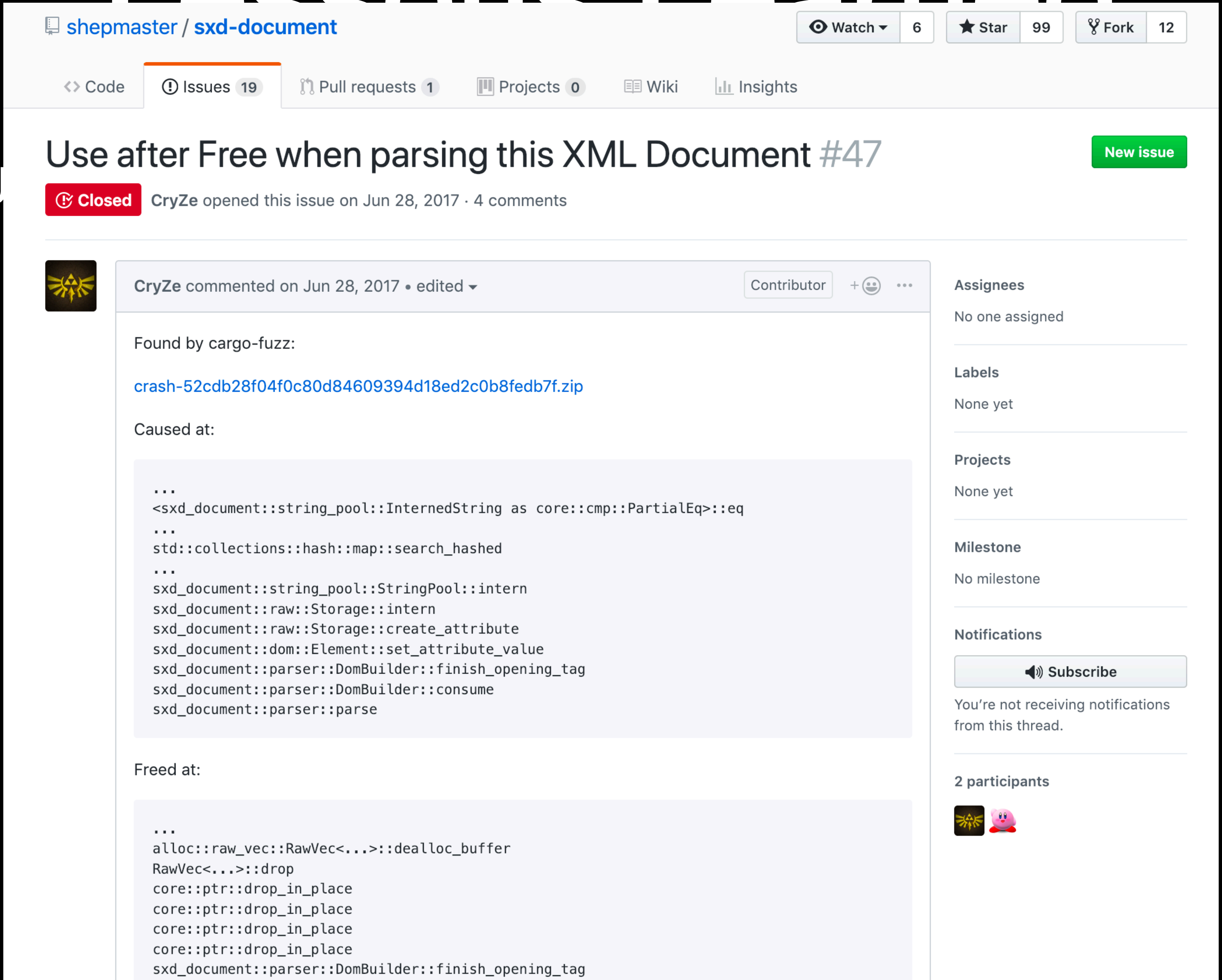
- **3,099** out of 10,693 Rust libraries (crates) contain unsafe code
- **14,796** files in total
- **651,193** lines of code

Lessons Learned

- Unsafe in the XML library

Lessons Learned

- U



Use after Free when parsing this XML Document #47

Closed **CryZe** opened this issue on Jun 28, 2017 · 4 comments

CryZe commented on Jun 28, 2017 • edited Contributor

Found by cargo-fuzz:

crash-52cdb28f04f0c80d84609394d18ed2c0b8fedb7f.zip

Caused at:

```
...
<sxd_document::string_pool::InternedString as core::cmp::PartialEq>::eq
...
std::collections::hash::map::search_hashed
...
sxd_document::string_pool::StringPool::intern
sxd_document::raw::Storage::intern
sxd_document::raw::Storage::create_attribute
sxd_document::dom::Element::set_attribute_value
sxd_document::parser::DomBuilder::finish_opening_tag
sxd_document::parser::DomBuilder::consume
sxd_document::parser::parse
```

Freed at:

```
...
alloc::raw_vec::RawVec<...>::dealloc_buffer
RawVec<...>::drop
core::ptr::drop_in_place
core::ptr::drop_in_place
core::ptr::drop_in_place
core::ptr::drop_in_place
sxd_document::parser::DomBuilder::finish_opening_tag
```

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

No milestone

Notifications

Subscribe

You're not receiving notifications from this thread.

2 participants

Lessons Learned

- Unsafe in the XML library
- CVEs in the Rust standard library

Lessons Learned

- Unsa
- CVEs

CVE-2018-1000657: **a Buffer Overflow vulnerability** in std::collections::vec_deque::VecDeque::reserve() function that can result in Arbitrary code execution

CVE-2018-1000810: The `str::repeat` function in the standard library allows repeating a string a fixed number of times, returning an owned version of the final string. The capacity of the final string is calculated by multiplying the length of the string being repeated by the number of copies. This calculation can overflow, and this case was not properly checked for.

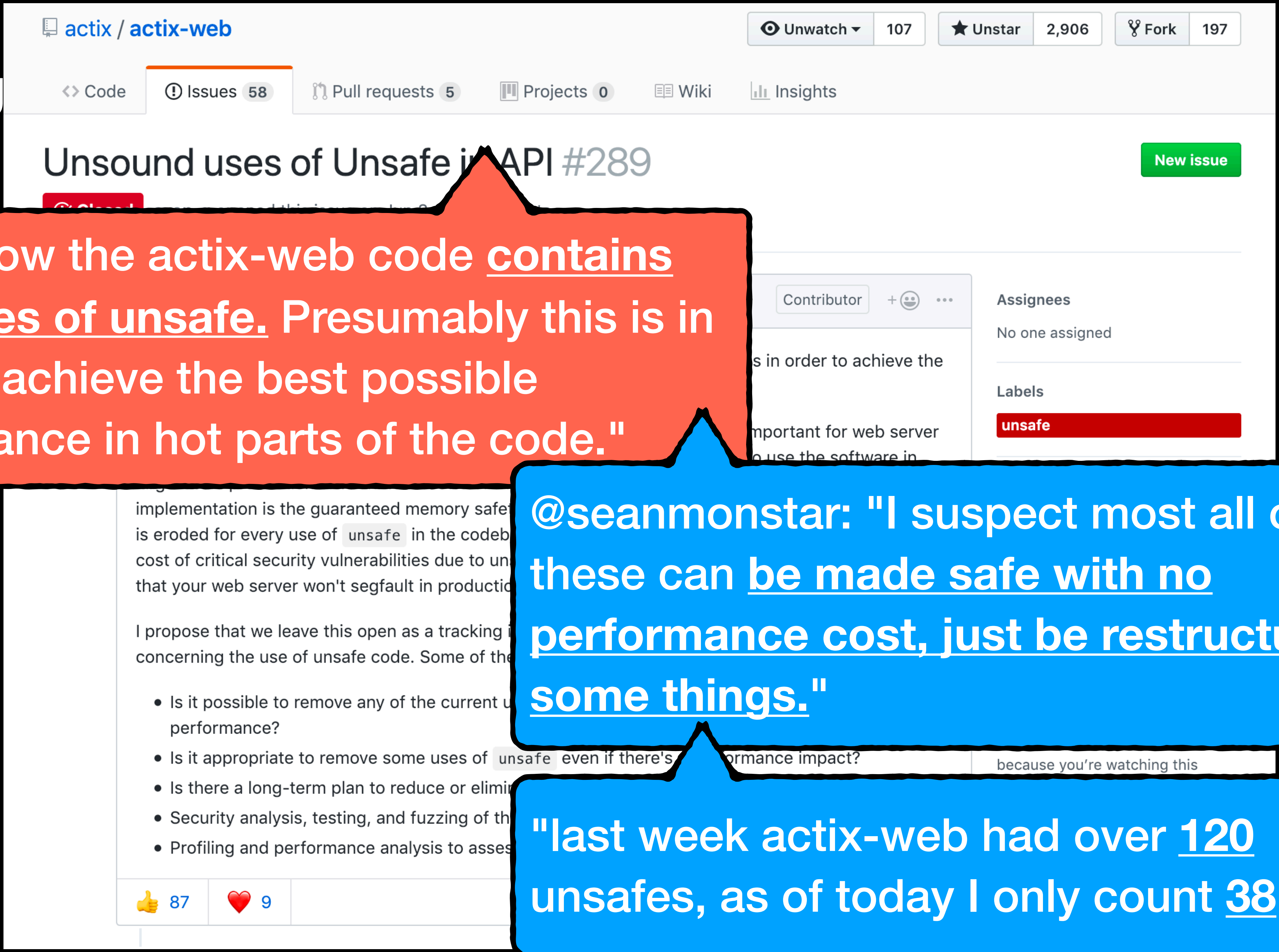
The rest of the implementation of `str::repeat` **contains unsafe code that relies on a preallocated vector having the capacity calculated earlier**. On integer overflow the capacity will be less than required, and which then writes outside of the allocated buffer, leading to buffer overflow.

Lessons Learned

- Unsafe in the XML library
- CVEs in the Rust standard library
- Unsafe in actix

Lessons Learned

- U



actix / actix-web

Unwatch 107 | Unstar 2,906 | Fork 197

Code | Issues 58 | Pull requests 5 | Projects 0 | Wiki | Insights

Unsound uses of Unsafe i API #289

New issue

Contributor

Assignees

No one assigned

Labels

unsafe

s in order to achieve the

mportant for web server

o use the software in

implementation is the guaranteed memory safe
is eroded for every use of `unsafe` in the codeb
cost of critical security vulnerabilities due to un
that your web server won't segfault in productio

I propose that we leave this open as a tracking i
concerning the use of unsafe code. Some of the

- Is it possible to remove any of the current u
performance?
- Is it appropriate to remove some uses of `unsafe` even if there's
rmance impact?
- Is there a long-term plan to reduce or elimi
- Security analysis, testing, and fuzzing of th
- Profiling and performance analysis to asses

👍 87 ❤️ 9

because you're watching this

"Right now the actix-web code contains 100+ uses of unsafe. Presumably this is in order to achieve the best possible performance in hot parts of the code."

@seanmonstar: "I suspect most all of these can be made safe with no performance cost, just be restructuring some things."

"last week actix-web had over 120 unsafes, as of today I only count 38."

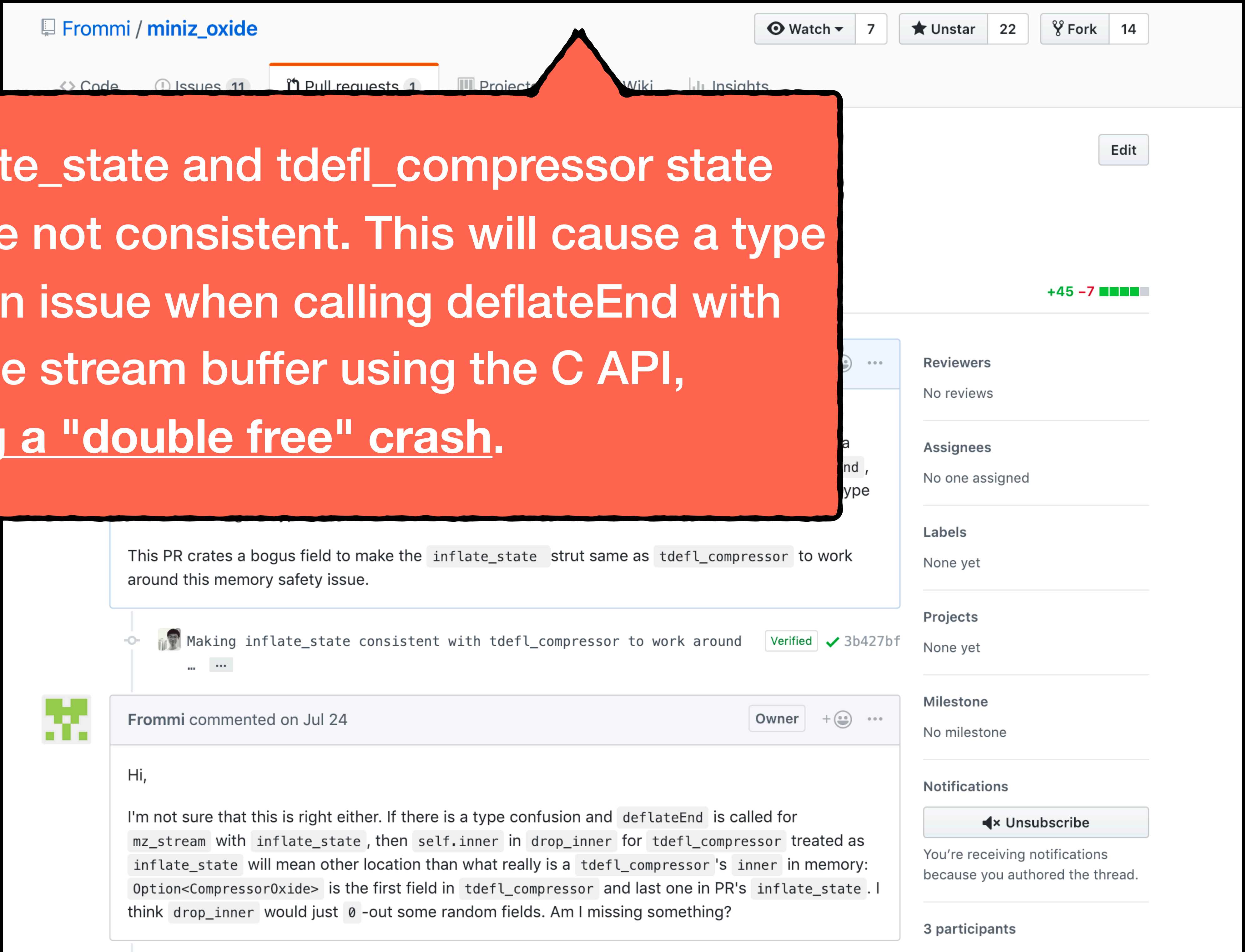
Lessons Learned

- Unsafe in the XML library
- CVEs in the Rust standard library
- Unsafe in actix
- FFI (Foreign Language Interface) in the miniz_oxide library

Lessons Learned

The inflate_state and tdefl_compressor state struct are not consistent. This will cause a type confusion issue when calling deflateEnd with the inflate stream buffer using the C API, resulting a "double free" crash.

rary



This PR crates a bogus field to make the `inflate_state` strut same as `tdefl_compressor` to work around this memory safety issue.

`Making inflate_state consistent with tdefl_compressor to work around …`

Frommi commented on Jul 24

Hi,

I'm not sure that this is right either. If there is a type confusion and `deflateEnd` is called for `mz_stream` with `inflate_state`, then `self.inner` in `drop_inner` for `tdefl_compressor` treated as `inflate_state` will mean other location than what really is a `tdefl_compressor`'s `inner` in memory: `Option<CompressorOxide>` is the first field in `tdefl_compressor` and last one in PR's `inflate_state`. I think `drop_inner` would just `0`-out some random fields. Am I missing something?

Reviewers
No reviews

Assignees
No one assigned

Labels
None yet

Projects
None yet

Milestone
No milestone

Notifications
You're receiving notifications because you authored the thread.

3 participants

Lessons Learned

- Unsafe in the XML library
- CVEs in the Rust standard library
- Unsafe in actix
- FFI (Foreign Language Interface) in the miniz_oxide library

How to Contribute?

- Rust Security Policy
- Google Groups (rustlang-security-announcements)
- RustSec Advisory Database
- Rust Secure Code Working Group
- The Rust Fuzz Project

Open Questions

- C to Rust **translation**
- Safety and security in the Rust **compiler** and **std**
- Unsafe Rust **code analysis**
- Rust unsafe code **sandbox** and **isolation**
- **Formal verification** of Rust and its libraries
- Memory-safety across various **boundaries**

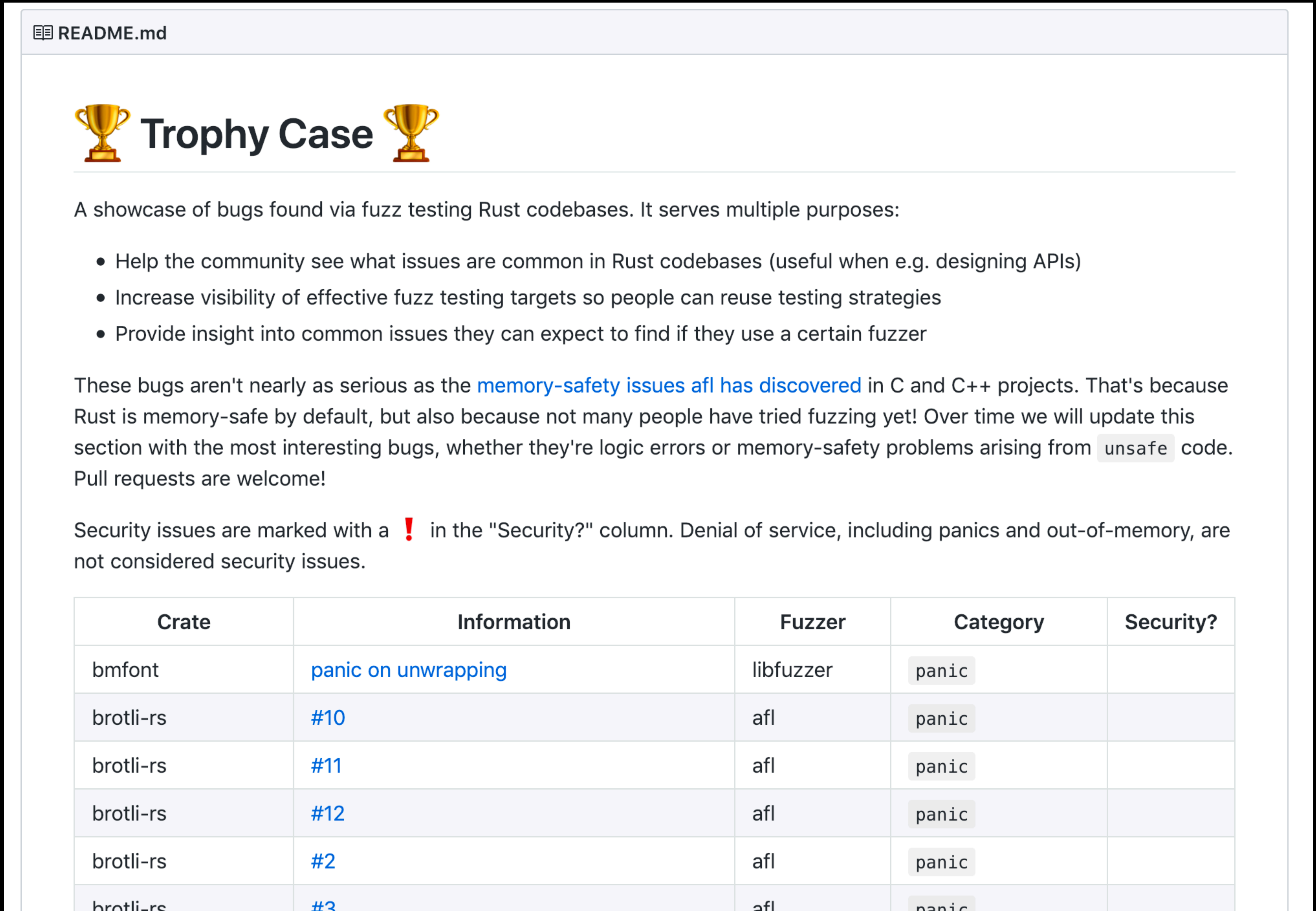
Existing Projects

- **Formal verification**
  - RustBelt: Securing the Foundations of the Rust Programming Language
  - Verifying Rust Programs with SMACK
  - RustSEM: An Operational Semantics for Rust Language
  - Other verification framework based on LLVM IR: SeaHorn

Existing Projects

- Formal verification
- **Fuzzing**
  - The Rust Fuzz Project: AFL, Hongfuzz, LLVM libFuzzer

Existing Projects



README.md

🏆 Trophy Case 🏆

A showcase of bugs found via fuzz testing Rust codebases. It serves multiple purposes:

- Help the community see what issues are common in Rust codebases (useful when e.g. designing APIs)
- Increase visibility of effective fuzz testing targets so people can reuse testing strategies
- Provide insight into common issues they can expect to find if they use a certain fuzzer

These bugs aren't nearly as serious as the memory-safety issues afl has discovered in C and C++ projects. That's because Rust is memory-safe by default, but also because not many people have tried fuzzing yet! Over time we will update this section with the most interesting bugs, whether they're logic errors or memory-safety problems arising from `unsafe` code. Pull requests are welcome!

Security issues are marked with a ❗ in the "Security?" column. Denial of service, including panics and out-of-memory, are not considered security issues.

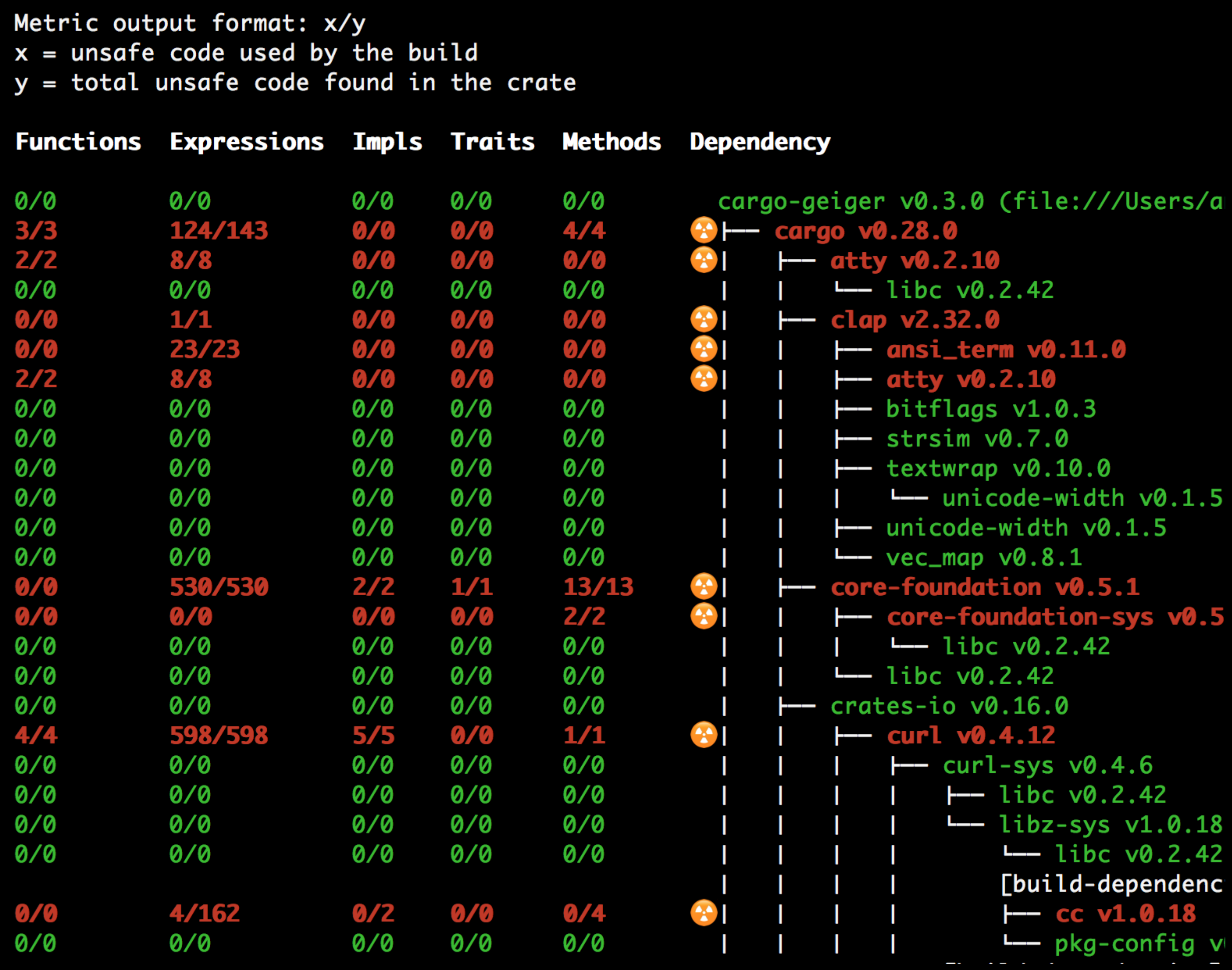
Crate	Information	Fuzzer	Category	Security?
bmfont	panic on unwrapping	libfuzzer	`panic`	
brotli-rs	#10	afl	`panic`	
brotli-rs	#11	afl	`panic`	
brotli-rs	#12	afl	`panic`	
brotli-rs	#2	afl	`panic`	
brotli-rs	#3	afl	`panic`	

Existing Projects

- Formal verification
- Fuzzing
- **Code analysis**
    - **Miri**: an experimental interpreter for Rust's mid-level intermediate representation (MIR).
        - out-of-bounds memory accesses and use-after-free
        - invalid use of uninitialized data
        - Violation of intrinsic preconditions
        - etc

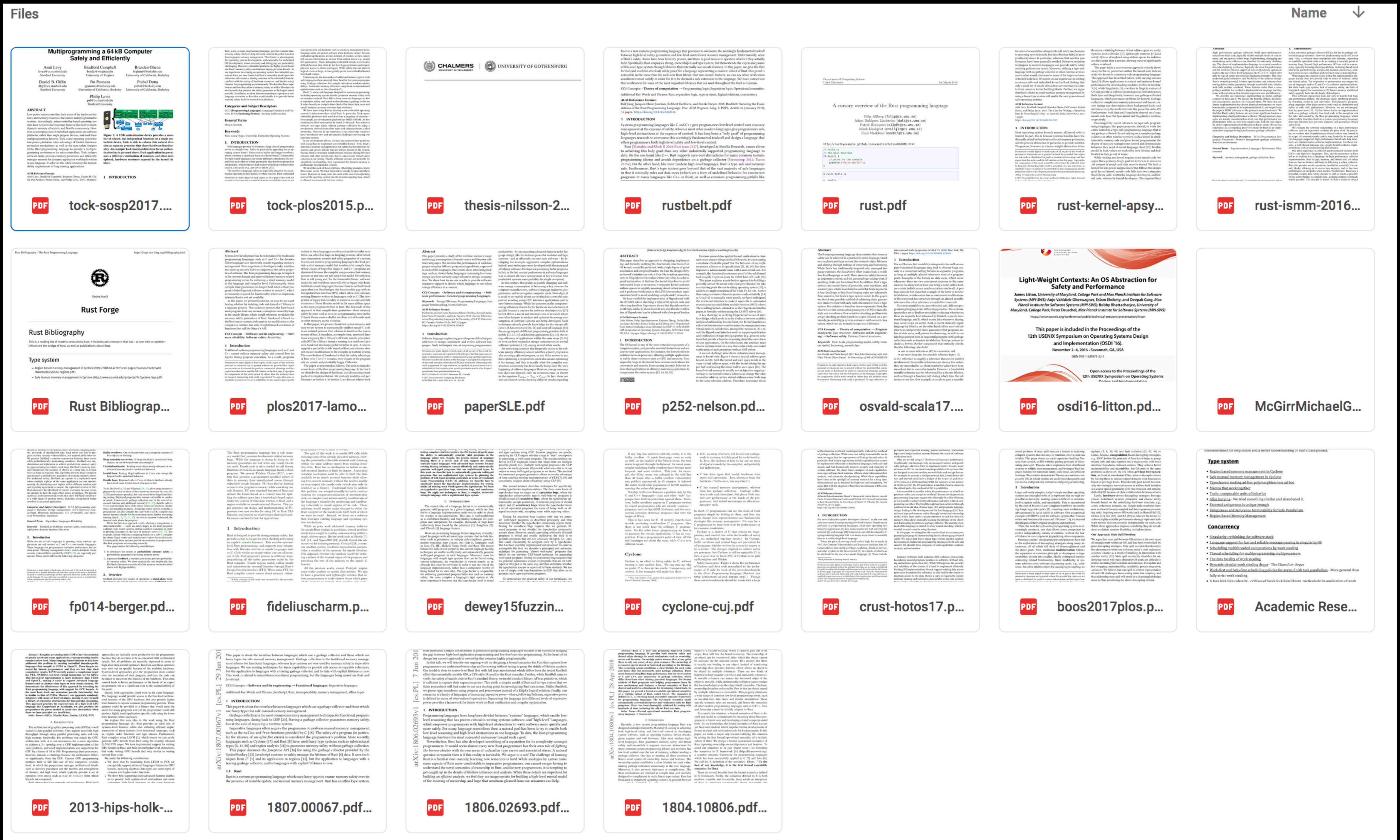
Existing Projects

- Formal verification
- Fuzzing
- Code analysis
- **Other tools**
  - `cargo geiger`



```
Metric output format: x/y
x = unsafe code used by the build
y = total unsafe code found in the crate

Functions  Expressions  Impls  Traits  Methods  Dependency

0/0        0/0          0/0    0/0     0/0        cargo-geiger v0.3.0 (file:///Users/a
3/3        124/143      0/0    0/0     4/4      ☢ ├── cargo v0.28.0
2/2        8/8          0/0    0/0     0/0      ☢ |   ├── atty v0.2.10
0/0        0/0          0/0    0/0     0/0        |   |   └── libc v0.2.42
0/0        1/1          0/0    0/0     0/0      ☢ |   ├── clap v2.32.0
0/0        23/23        0/0    0/0     0/0      ☢ |   |   ├── ansi_term v0.11.0
2/2        8/8          0/0    0/0     0/0      ☢ |   |   ├── atty v0.2.10
0/0        0/0          0/0    0/0     0/0        |   |   ├── bitflags v1.0.3
0/0        0/0          0/0    0/0     0/0        |   |   ├── strsim v0.7.0
0/0        0/0          0/0    0/0     0/0        |   |   ├── textwrap v0.10.0
0/0        0/0          0/0    0/0     0/0        |   |   |   └── unicode-width v0.1.5
0/0        0/0          0/0    0/0     0/0        |   |   ├── unicode-width v0.1.5
0/0        0/0          0/0    0/0     0/0        |   |   └── vec_map v0.8.1
0/0        530/530      2/2    1/1     13/13    ☢ |   ├── core-foundation v0.5.1
0/0        0/0          0/0    0/0     2/2      ☢ |   |   ├── core-foundation-sys v0.5
0/0        0/0          0/0    0/0     0/0        |   |   |   └── libc v0.2.42
0/0        0/0          0/0    0/0     0/0        |   |   └── libc v0.2.42
0/0        0/0          0/0    0/0     0/0        |   ├── crates-io v0.16.0
4/4        598/598      5/5    0/0     1/1      ☢ |   |   ├── curl v0.4.12
0/0        0/0          0/0    0/0     0/0        |   |   |   ├── curl-sys v0.4.6
0/0        0/0          0/0    0/0     0/0        |   |   |   |   ├── libc v0.2.42
0/0        0/0          0/0    0/0     0/0        |   |   |   |   └── libz-sys v1.0.18
0/0        0/0          0/0    0/0     0/0        |   |   |   |       └── libc v0.2.42
                                                  |   |   |   |       [build-dependenc
0/0        4/162        0/2    0/0     0/4      ☢ |   |   |   |       ├── cc v1.0.18
0/0        0/0          0/0    0/0     0/0        |   |   |   |       └── pkg-config v
```

Papers https://goo.gl/99Rg3c



Files

Name ↓

PDF tock-sosp2017....

PDF tock-plos2015.p...

PDF thesis-nilsson-2...

PDF rustbelt.pdf

PDF rust.pdf

PDF rust-kernel-apsy...

PDF rust-ismm-2016...

PDF Rust Bibliograp...

PDF plos2017-lamo...

PDF paperSLE.pdf

PDF p252-nelson.pd...

PDF osvald-scala17....

PDF osdi16-litton.pd...

PDF McGirrMichaelG...

PDF fp014-berger.pd...

PDF fideliuscharm.p...

PDF dewey15fuzzin...

PDF cyclone-cuj.pdf

PDF crust-hotos17.p...

PDF boos2017plos.p...

PDF Academic Rese...

PDF 2013-hips-holk-...

PDF 1807.00067.pdf...

PDF 1806.02693.pdf...

PDF 1804.10806.pdf...

Conclusion

- Building safe and secure systems in Rust
- Challenges, lessons learned, and open questions

Questions?