Exploiting use after free and double free in Rust stdlib

CVE-2020-36318

Obianuju Chika-Nwanja

Jasmine Yew

Yi Cai

Shubham Kulkarni

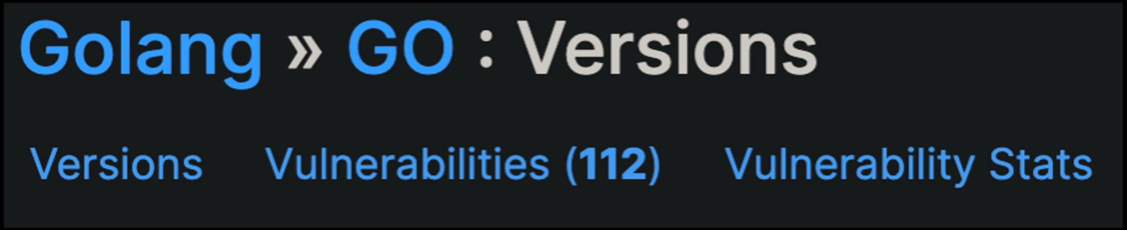
Kaiyang Yu

Agenda

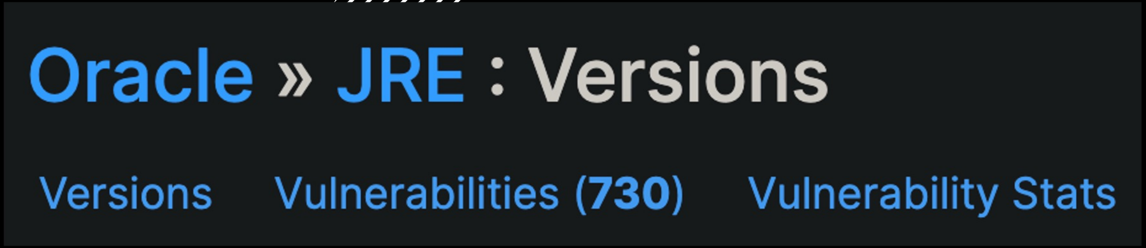
- Rust
- Vulnerability
- Demo
- The Fix
- Why is this important?
- Conclusion

What is Rust?

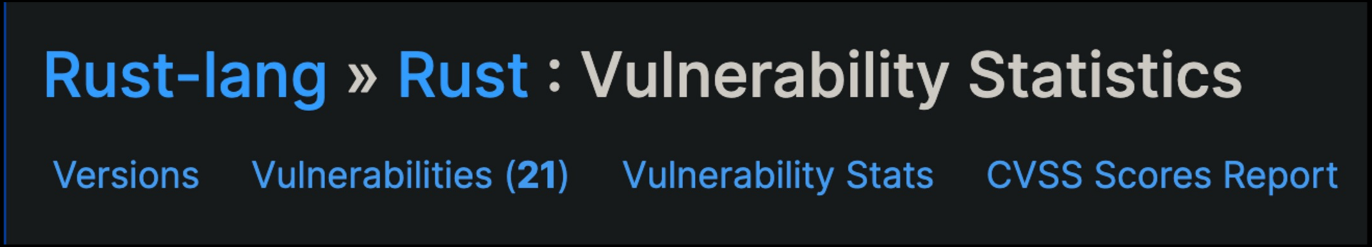
- Systems programming language
- First stable release in 2015
- Maintained now by Rust Foundation
- Known for
  - No null pointers
  - Ownership and borrowing system
  - **MEMORY SAFETY**


Golang » GO : Versions

Versions Vulnerabilities (112) Vulnerability Stats


Oracle » JRE : Versions

Versions Vulnerabilities (730) Vulnerability Stats


Rust-lang » Rust : Vulnerability Statistics

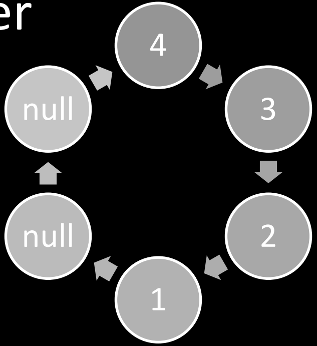
Versions Vulnerabilities (21) Vulnerability Stats CVSS Scores Report

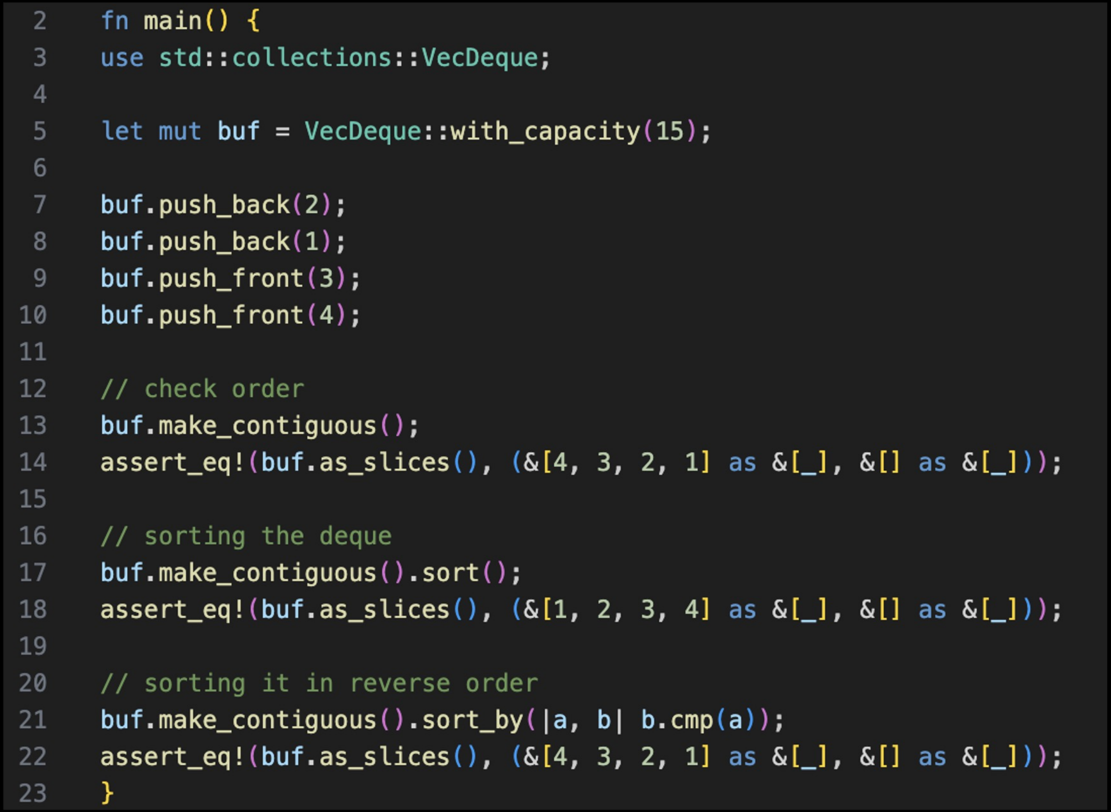
Rust – Memory Safe ?

- Memory-related vulnerabilities like :
  - Buffer overflow - **do I need to explain?**
  - Use after free - referencing memory after freed
  - Double free - calling free twice on the same memory address

VecDeque

- Vuln located in make_contiguous()
- VecDeque is a double ended queue implemented with a ring buffer

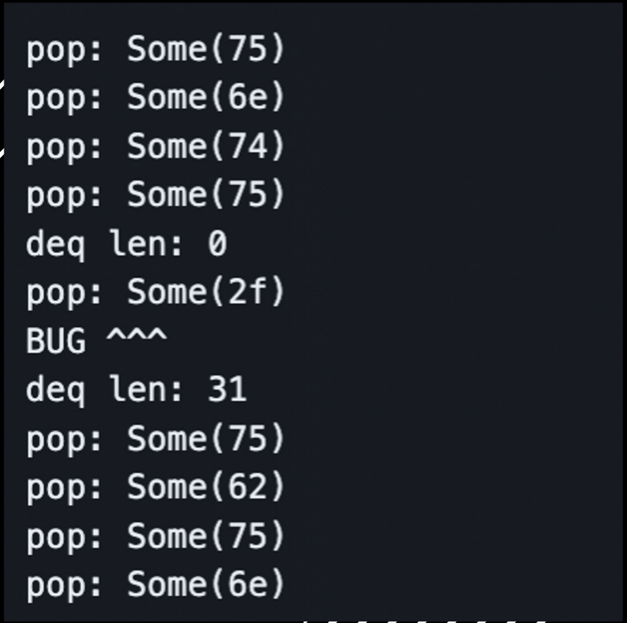




```
fn main() {
use std::collections::VecDeque;

let mut buf = VecDeque::with_capacity(15);

buf.push_back(2);
buf.push_back(1);
buf.push_front(3);
buf.push_front(4);

// check order
buf.make_contiguous();
assert_eq!(buf.as_slices(), (&[4, 3, 2, 1] as &[_], &[] as &[_]));

// sorting the deque
buf.make_contiguous().sort();
assert_eq!(buf.as_slices(), (&[1, 2, 3, 4] as &[_], &[] as &[_]));

// sorting it in reverse order
buf.make_contiguous().sort_by(|a, b| b.cmp(a));
assert_eq!(buf.as_slices(), (&[4, 3, 2, 1] as &[_], &[] as &[_]));
}
```

Original code from rust-lang docs on make_contiguous

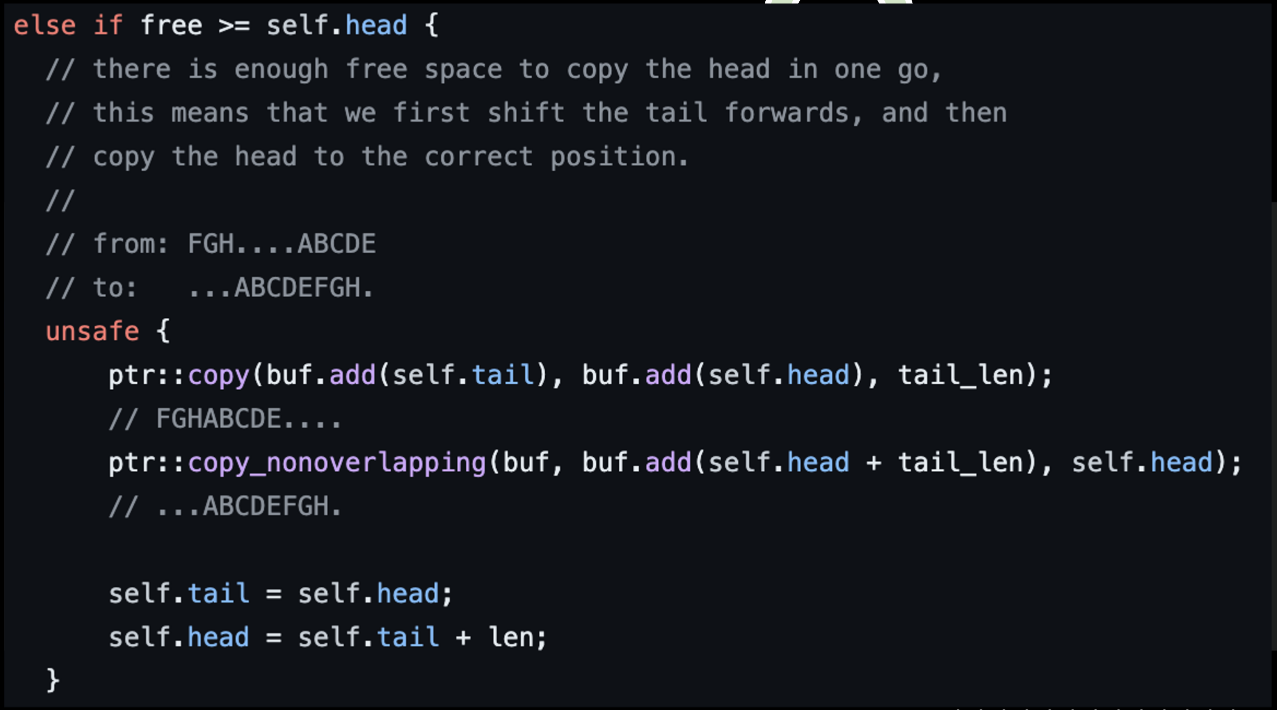
Vulnerability Realization



```
pop: Some(75)
pop: Some(6e)
pop: Some(74)
pop: Some(75)
deq len: 0
pop: Some(2f)
BUG ^^^
deq len: 31
pop: Some(75)
pop: Some(62)
pop: Some(75)
pop: Some(6e)
```

- Discovered by Andrew Yourtchenko while developing a simple SFTP server
- Rust Version 1.48.0 (commit 7eac88abb 2020-11-16)
- Issue created on GitHub on Dec. 7, 2020

Vulnerable Code



```
else if free >= self.head {
  // there is enough free space to copy the head in one go,
  // this means that we first shift the tail forwards, and then
  // copy the head to the correct position.
  //
  // from: FGH....ABCDE
  // to:   ...ABCDEFGH.
  unsafe {
      ptr::copy(buf.add(self.tail), buf.add(self.head), tail_len);
      // FGHABCDE....
      ptr::copy_nonoverlapping(buf, buf.add(self.head + tail_len), self.head);
      // ...ABCDEFGH.

      self.tail = self.head;
      self.head = self.tail + len;
  }
```

Rust v1.48.0 code snippet of make_contiguous and is_contiguous

The Vulnerability

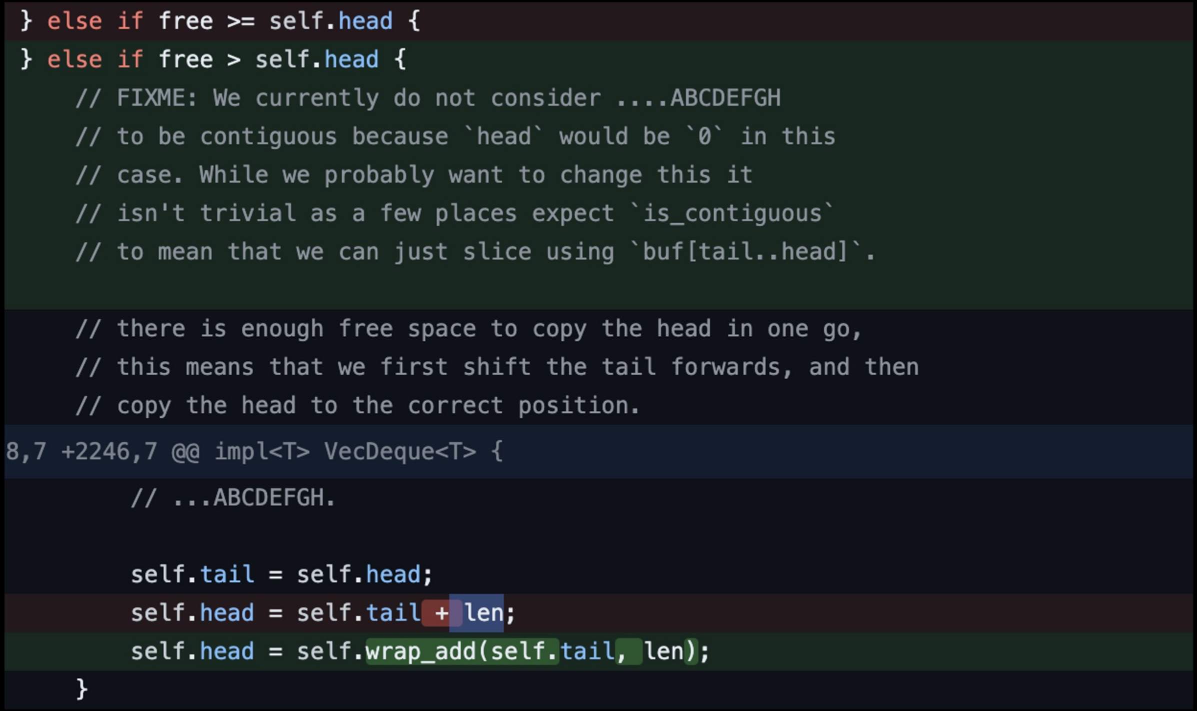
- CVE-2020-36318
- CVSS Score: 9.8
- Popped the same elements more than once
  - Double free
  - Use-after-free

Demo

The Fix

- The issue was published on Dec 7, 2020 (https://github.com/rust-lang/rust/issues/79808)
- The issue was fixed on the same day, and merged three days later (https://github.com/rust-lang/rust/pull/79814)
- The total timeline was very short

The Fix



```
} else if free >= self.head {
} else if free > self.head {
    // FIXME: We currently do not consider ....ABCDEFGH
    // to be contiguous because `head` would be `0` in this
    // case. While we probably want to change this it
    // isn't trivial as a few places expect `is_contiguous`
    // to mean that we can just slice using `buf[tail..head]`.

    // there is enough free space to copy the head in one go,
    // this means that we first shift the tail forwards, and then
    // copy the head to the correct position.
8,7 +2246,7 @@ impl<T> VecDeque<T> {
        // ...ABCDEFGH.

        self.tail = self.head;
        self.head = self.tail + len;
        self.head = self.wrap_add(self.tail, len);
    }
```

Why is this important?

- Memory unsafe
- Undefined behavior
- Attackers can exploit this to
  - Make programs crash
  - Execute code
- Examples:
  - CVE-2021-0920 – Android kernel
  - CVE-2020-6819 – Thunderbird and Firefox
  - CVE-2006-5051 – OpenSSH

Takeaways

- Rust is as safe as the implementation goes.
- The Rust standard library uses unsafe{} code blocks for its implementation, which doesn't have memory-safe guarantee.
- Extensive testing and code reviews are important for standard library implementation.
- Even though Rust is memory safe in most cases, it's not completely safe.

References

[1] “VecDeque: length 0 underflow and bogus values from pop_front(), triggered by a certain sequence of reserve(), push_back(), make_contiguous(), pop_front() · Issue #79808 · rust-lang/rust,” *GitHub*. https://github.com/rust-lang/rust/issues/79808

[2] “fix soundness issue in `make_contiguous` by lcnr · Pull Request #79814 · rust-lang/rust,” *GitHub*. https://github.com/rust-lang/rust/pull/79814/files#diff-47b09db89738e45a04cc9fb1f000075f21c1f59f91e642f7b4d89857ac7f7c31

[3] “rust/library/alloc/src/collections/vec_deque.rs at 7eac88abb2e57e752f3302f02be5f3ce3d7adfb4 · rust-lang/rust,” *GitHub*. https://github.com/rust-lang/rust/blob/7eac88abb2e57e752f3302f02be5f3ce3d7adfb4/library/alloc/src/collections/vec_deque.rs

References

[4] “VecDeque in std::collections - Rust,” *Rust*. https://doc.rust-lang.org/std/collections/struct.VecDeque.html#method.make_contiguous

[5] “Introduction - The Rust Programming Language,” *Rust*. https://doc.rust-lang.org/book/ch00-00-introduction.html

[6] “you’re probably infringing the Rust trademark by using their logo · Issue #55 · crablang/crab,” *GitHub*. https://github.com/crablang/crab/issues/55

[7] “NVD - CVE-2020-36318,” *National Vulnerability Database*. https://nvd.nist.gov/vuln/detail/CVE-2020-36318#VulnChangeHistorySection

Questions