Is rust language really safe?

github.com/jrabello

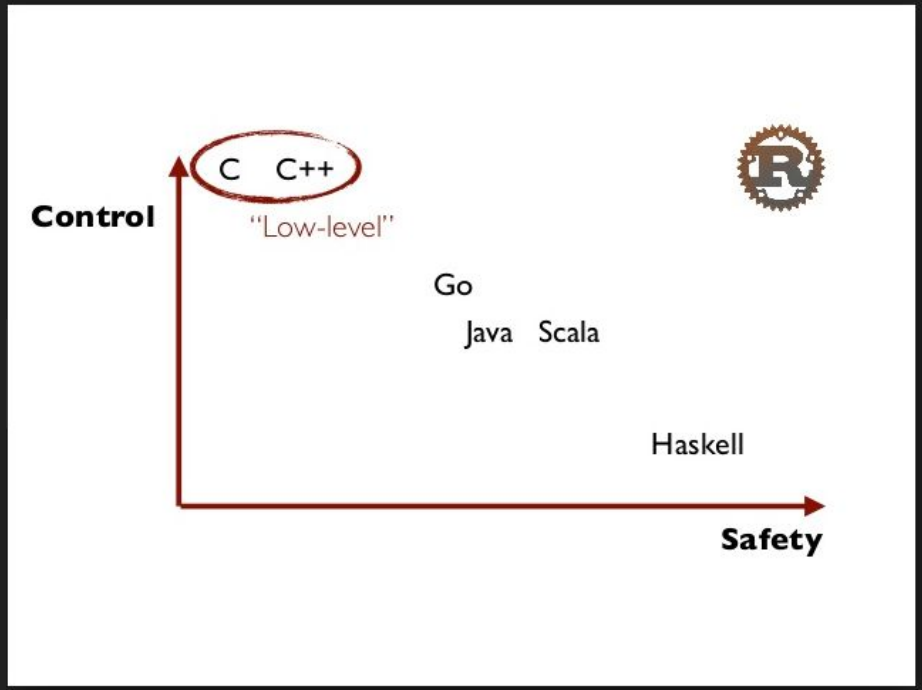
Definition of safe

“Safety” é o estado de estar "seguro" (do francês sauf), a condição de ser protegido de danos ou resultados não desejáveis. [3]

Why safety in software matters?

- Em janeiro de 2009, o mecanismo de busca do Google erroneamente notificou os usuários de que todos os sites em todo o mundo eram potencialmente maliciosos, incluindo o seu.[1]
- Um bug no código que controlava a máquina de terapia por radiação Therac-25 foi diretamente responsável por pelo menos cinco mortes de pacientes nos anos 80, quando administrou quantidades excessivas de radiação beta.[1]
- O Ariane 5 Flight 501 da Agência Espacial Européia foi destruído 40 segundos após a decolagem (4 de junho de 1996). O protótipo de foguete de US$ 1 bilhão foi destruído devido a um bug no software de orientação a bordo.[1]
- Knight's $440 Million Error – Um bug de software(high frequency trading) provocou uma perda de US$ 440 milhões em apenas 30 minutos.[1]
- Falhas em software custaram um total de 1.7 trilhoes de dolares em 2017.[8]

Safety, Control and Performance

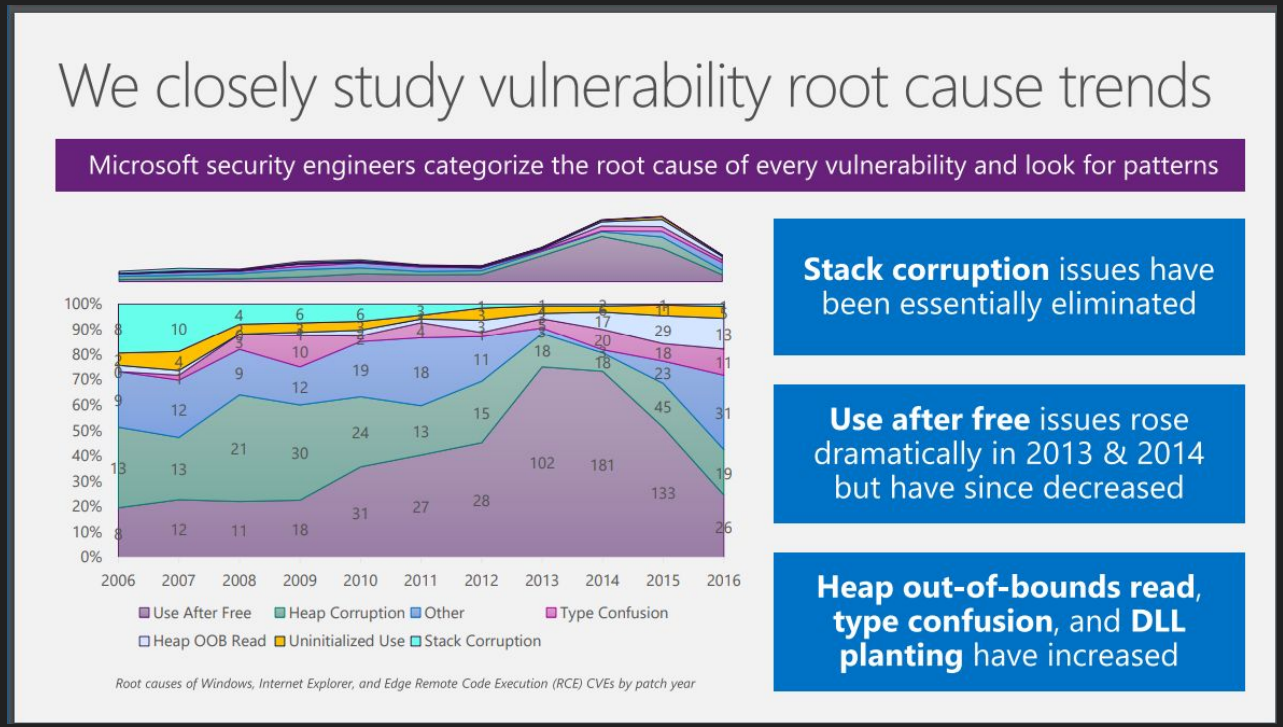


Rust Memory Safety Model

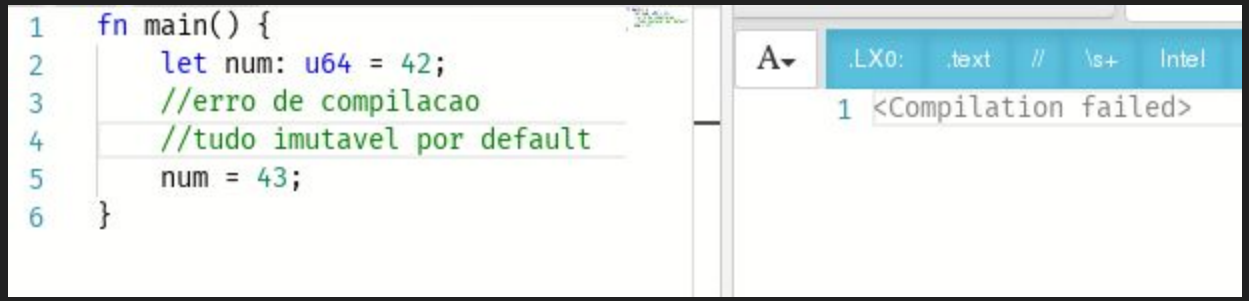
- No null pointer dereferences
- No dangling pointers
- No Iterator Invalidation
- No use after free
- No double frees
- No out of bound access
- No buffer overflows
- No data races

Vulnerabilities Map

- 2016, 56% dos bugs Heap Corruption.[9]



Rust Default Immutability



Rust Default Immutability

```rust
fn main() {
    let num: u64 = 42;
    //erro de compilacao
    //tudo imutavel por default
    num = 43;
}
```

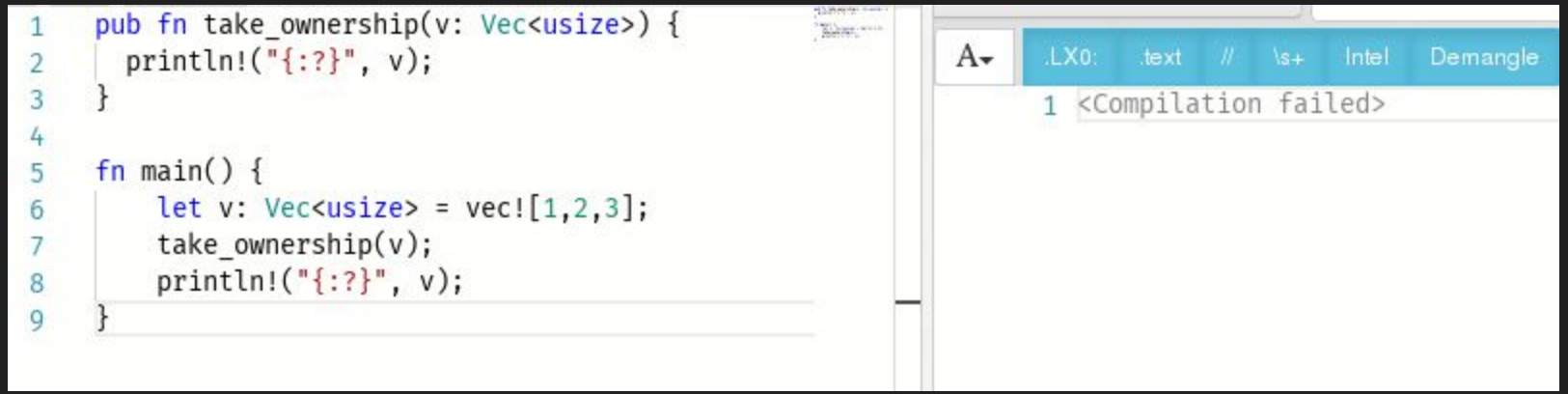
```
<Compilation failed>
```

```rust
fn main() {
    let mut num: u64 = 42;
    num = 43;
}
```

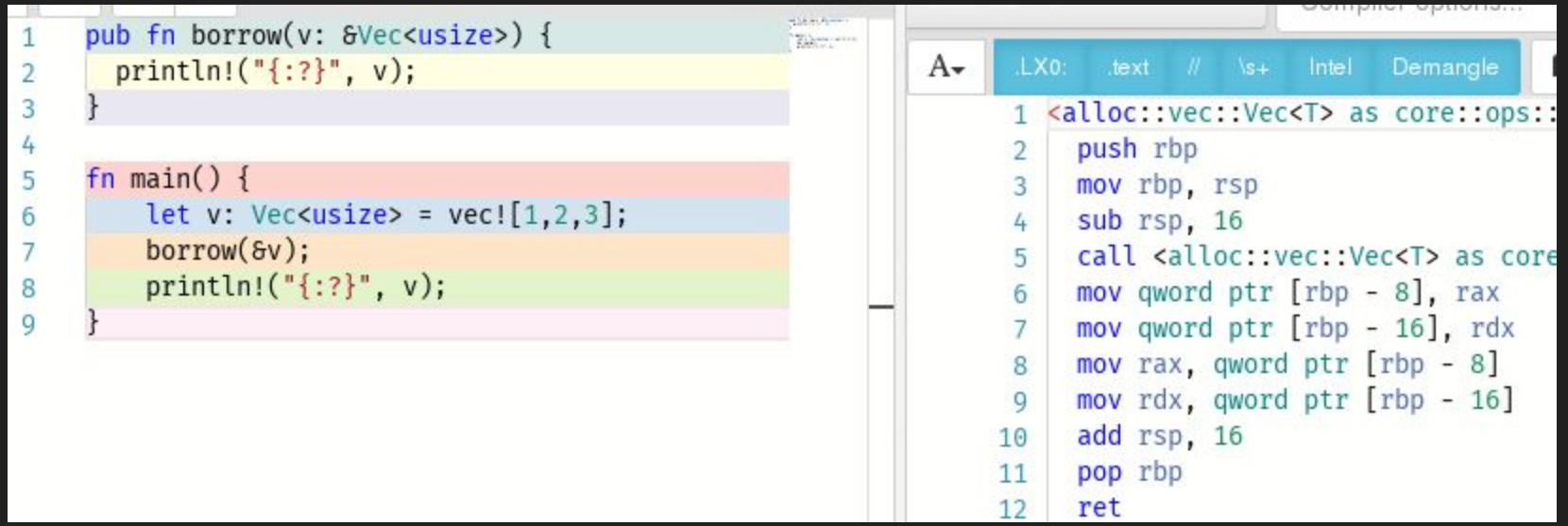
```
example::main:
  push rbp
  mov rbp, rsp
  push rax
  mov qword ptr [rbp - 8], 42
  mov qword ptr [rbp - 8], 43
  add rsp, 8
  pop rbp
  ret
```

Rust Ownership

“Prevents use after free in compile time...”[6]



Rust Borrowing



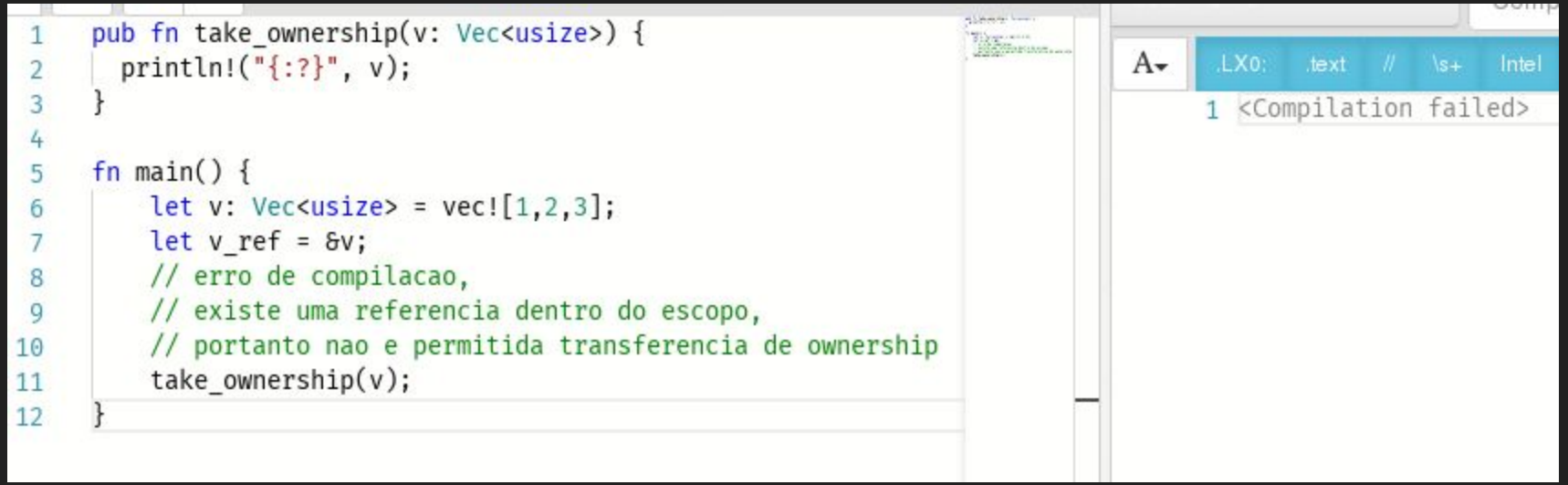
```rust
pub fn borrow(v: &Vec<usize>) {
  println!("{:?}", v);
}

fn main() {
    let v: Vec<usize> = vec![1,2,3];
    borrow(&v);
    println!("{:?}", v);
}
```

```
<alloc::vec::Vec<T> as core::ops::
  push rbp
  mov rbp, rsp
  sub rsp, 16
  call <alloc::vec::Vec<T> as core
  mov qword ptr [rbp - 8], rax
  mov qword ptr [rbp - 16], rdx
  mov rax, qword ptr [rbp - 8]
  mov rdx, qword ptr [rbp - 16]
  add rsp, 16
  pop rbp
  ret
```

Rust Borrowing

“Borrowing prevents moving...”[6]



```rust
pub fn take_ownership(v: Vec<usize>) {
  println!("{:?}", v);
}

fn main() {
    let v: Vec<usize> = vec![1,2,3];
    let v_ref = &v;
    // erro de compilacao,
    // existe uma referencia dentro do escopo,
    // portanto nao e permitida transferencia de ownership
    take_ownership(v);
}
```

```
<Compilation failed>
```

Stack Based Memory Corruption 1

```
1  // Type your code here, or load an example.
2  fn evil() {
3    let mut arr = [0xff;0xff];
4    arr[0x999] = 0x7f;
5  }
6
```

```
22 .LBB0_3:
23   mov rax, qword ptr [rbp - 1040]
24   mov rcx, qword ptr [rbp - 1032]
25   cmp rax, rcx
26   mov qword ptr [rbp - 1048], rax
27   jne .LBB0_2
28   xor eax, eax
29   mov cl, al
30   test cl, 1
31   jne .LBB0_1
32   lea rax, [rip + panic_bounds_check_loc.1]
33   mov ecx, 2457
34   mov esi, ecx
35   mov ecx, 255
36   mov edx, ecx
37   mov rdi, rax
38   call core::panicking::panic_bounds_check@PLT
```

A▾ | .LX0: | .text | // | \s+ | Intel | Demangle

```
1  example::evil:
2    push rbp
3    mov rbp, rsp
4    sub rsp, 1056
5    lea rax, [rbp - 1020]
6    mov rcx, rax
7    add rcx, 1020
8    mov qword ptr [rbp - 1032], rcx
9    mov qword ptr [rbp - 1040], rax
10   jmp .LBB0_3
11 .LBB0_1:
12   mov dword ptr [rbp + 8808], 127
13   add rsp, 1056
14   pop rbp
15   ret
16 .LBB0_2:
17   mov rax, qword ptr [rbp - 1048]
18   add rax, 4
19   mov rcx, qword ptr [rbp - 1048]
20   mov dword ptr [rcx], 255
21   mov qword ptr [rbp - 1040], rax
22 .LBB0_3:
23   mov rax, qword ptr [rbp - 1040]
24   mov rcx, qword ptr [rbp - 1032]
25   cmp rax, rcx
```

rustc 1.21.0 (3b72af97e 2017-10-09) - *cached*

Stack Based Memory Corruption 1

```
jrabelo@zion ~/codes/rust/nullbyte/nullbyte (master*) $ cargo run
   Compiling nullbyte v0.1.0 (file:///home/jrabelo/codes/rust/nullbyte/nullbyte)
warning: this expression will panic at run-time
 --> src/main.rs:3:3
  |
3 |   arr[0x999] = 0x7f;
  |   ^^^^^^^^^^ index out of bounds: the len is 255 but the index is 2457

    Finished dev [unoptimized + debuginfo] target(s) in 0.36 secs
     Running `target/debug/nullbyte`
thread 'main' panicked at 'index out of bounds: the len is 255 but the index is 2457', src/main.rs:3:3
note: Run with `RUST_BACKTRACE=1` for a backtrace.
jrabelo@zion ~/codes/rust/nullbyte/nullbyte (master*) $
```

Stack Based Memory Corruption 2

```rust
// Type your code here, or load an example.
fn evil(nice_idx: usize) {
  let mut arr = [0xff;0xff];
  arr[nice_idx] = 0x7f;
}
```

```
example::evil:
  push rbp
  mov rbp, rsp
  sub rsp, 1056
  lea rax, [rbp - 1020]
  mov rcx, rax
  add rcx, 1020
  mov qword ptr [rbp - 1032], rdi
  mov qword ptr [rbp - 1040], rcx
  mov qword ptr [rbp - 1048], rax
  jmp .LBB0_3
.LBB0_1:
  mov rax, qword ptr [rbp - 1032]
  mov dword ptr [rbp + 4*rax - 1020], 127
  add rsp, 1056
  pop rbp
  ret
.LBB0_2:
  mov rax, qword ptr [rbp - 1056]
  add rax, 4
```

```
jrabelo@zion ~/codes/rust/nullbyte/nullbyte (master*) $ cargo run
    Finished dev [unoptimized + debuginfo] target(s) in 0.0 secs
     Running `target/debug/nullbyte`
thread 'main' panicked at 'index out of bounds: the len is 255 but the index is 2457', src/main.rs:3:3
note: Run with `RUST_BACKTRACE=1` for a backtrace.
jrabelo@zion ~/codes/rust/nullbyte/nullbyte (master*) $
```

Heap Based Memory Corruption

```
1  fn evil_vec(nice_idx: usize) {
2      let mut list = vec![1,2,3];
3      list[nice_idx] = 0x7f;
4  }
5
6  fn main() {
7      evil_vec(0x999);
8  }
```

```
1055   mov rsi, rcx
1056   call alloc::slice::<impl
1057   jmp .LBB46_2
1058 .LBB46_1:
1059   mov eax, dword ptr [rbp
1060   mov rdi, qword ptr [rbp
1061   mov dword ptr [rbp - 52], eax
1062   call _Unwind_Resume@PLT
1063 .LBB46_2:
1064   lea rdi, [rbp - 40]
1065   mov rsi, qword ptr [rbp - 48]
1066   call <alloc::vec::Vec<T> as core::ops::index::IndexMut<usize>>::index_mut
1067   mov qword ptr [rbp - 64], rax
1068   jmp .LBB46_3
1069 .LBB46_3:
1070   lea rdi, [rbp - 40]
1071   mov rax, qword ptr [rbp - 64]
1072   mov dword ptr [rax], 127
1073   call core::ptr::drop_in_place
1074   jmp .LBB46_5
1075 .LBB46_4:
1076   lea rdi, [rbp - 40]
1077   call core::ptr::drop_in_place
1078   jmp .LBB46_1
```

```
.LBB45_3:
  lea rax, [rip + panic_bounds_check_loc.1]
  mov rdi, rax
  mov rsi, qword ptr [rbp - 8]
  mov rdx, qword ptr [rbp - 24]
  call core::panicking::panic_bounds_check@PLT
```

rustc 1.21.0 (3b72af97e 2017-10-09) 1077ms

Unsafe keyword

“...problemas devido a falta de segurança de memória como null pointer dereference podem ocorrer.”[5]

Unsafe Memory Corruption

Vector set_len idea[7]

```
1  pub fn evil() {
2      let mut v: Vec<usize> = vec![1,2,3];
3      unsafe {
4          v.set_len(99999);
5      }
6      v[99998] = 0xccccccccccccccc;
7  }
8
9  fn main() {
10     evil();
11 }
```

```
1065   mov rdi, qword ptr [rbp - 16]
1066   mov dword ptr [rbp - 44], eax
1067   call _Unwind_Resume@PLT
1068 .LBB47_2:
1069   mov eax, 99999
1070   mov esi, eax
1071   lea rdi, [rbp - 40]
1072   call <alloc::vec::Vec<T>>::set_len
1073   jmp .LBB47_3
1074 .LBB47_3:
1075   mov eax, 99998
1076   mov esi, eax
1077   lea rdi, [rbp - 40]
1078   call <alloc::vec::Vec<T> as core::ops::index::IndexMut<usize>>::index_mut
1079   mov qword ptr [rbp - 56], rax
1080   jmp .LBB47_6
1081 .LBB47_4:
1082   lea rdi, [rbp - 40]
1083   call core::ptr::drop_in_place
1084   jmp .LBB47_1
1085 .LBB47_5:
1086   jmp .LBB47_4
1087 .LBB47_6:
1088   lea rdi, [rbp - 40]
1089   movabs rax, -3689348814741910324
1090   mov rcx, qword ptr [rbp - 56]
```

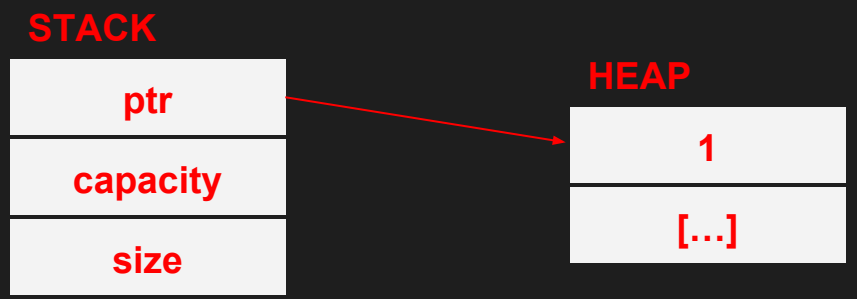
rustc 1.21.0 (3b72af97e 2017-10-09) - 1242ms

Unsafe Memory Corruption

```
(gdb) r
Starting program: /home/jrabelo/codes/rust/nullbyte/nullbyte/src/main
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Program received signal SIGSEGV, Segmentation fault.
0x000055555555bdfd in main::evil () at ./main.rs:8
8           v[0x99998] = 0xcccccccccccccccc;
(gdb) p &v
$1 = (alloc::vec::Vec<usize> *) 0x7fffffffdc48
(gdb) x/8gx 0x7fffffffdc48
0x7fffffffdc48: 0x00007ffff6c20060      0x0000000000000003
0x7fffffffdc58: 0x0000000000099999      0x0000000000000018
0x7fffffffdc68: 0x000055555556c133      0x00007fffffffdc80
0x7fffffffdc78: 0x000055555555be39      0x00007fffffffdce0
(gdb) x/x 0x00007ffff6c20060+(0x99998*8)
0x7ffff70ecd20: Cannot access memory at address 0x7ffff70ecd20
(gdb) info reg
rax            0xcccccccccccccccc       -3689348814741910324
rbx            0x0      0
rcx            0x7ffff70ecd20   140737338330400
rdx            0x99901  628993
rsi            0x7ffff6c20060   140737333297248
rdi            0x7fffffffdc48   140737488346184
rbp            0x7fffffffdc70   0x7fffffffdc70
rsp            0x7fffffffdc30   0x7fffffffdc30
r8             0x7fffffffdb20   140737488345888
r9             0x55555559ed80   93824992537984
r10            0x7ffff6c0f248   140737333228104
```

Capacity WTF??



Conclusion

- O Rust compiler realiza inferencias em tempo de compilacao, porem nos exemplos que foram demonstrados aqui, algumas checagens feitas em runtime poderiam ter sido feitos em tempo de compilacao
- Rust apresenta novos desafios para pesquisadores de seguranca da informacao
- Exploracao de vulnerabilidades em software escrito em Rust totalmente plausivel mas nao trivial

References

1. https://en.wikipedia.org/wiki/List_of_software_bugs
2. https://www.exploit-db.com/exploit-database-statistics/
3. https://en.wikipedia.org/wiki/Safety
4. https://www.rust-lang.org/en-US/
5. https://doc.rust-lang.org/book/second-edition/ch19-01-unsafe-rust.html
6. https://www.youtube.com/watch?v=agzf6ftEsLU
7. https://avadacatavra.github.io/rust/gdb/exploit/2017/09/26/attackingrustforfunandprofit.html
8. https://www.techrepublic.com/article/report-software-failure-caused-1-7-trillion-in-financial-losses-in-2017/
9. https://www.blackhat.com/docs/us-16/materials/us-16-Weston-Windows-10-Mitigation-Improvements.pdf