MONTREHACK | RUDIMENTARY RUST REVERSING

ALEXANDRE BEAULIEU (@ALXBL)

FEB. 19, 2020

RUST FUNDAMENTALS

CALLING CONVENTION (LINUX X64)

- Similar to **`C`**
- Frist 6 arguments in registers
  - **`rdi, rsi, rdx, rcx, r8, r9`**
- Rest on stack
- Return value in **`rax`**
- Caller frees arguments

ADDRESSING

- All arguments are addressed **`rsp`** relative.
  - Depends on stack size (**`sub rsp, <N>`**)
- Code is Position Independent
  - **`PLT`** calls are **`rip`**-relative
  - **`GOT`** offsets are **`rip`**-relative

CALLER EXAMPLE

```
example::main:
  sub rsp, 24 ; Allocate 24 bytes for arguments
  mov edi, 1
  mov esi, 2
  mov edx, 3
  mov ecx, 4
  mov r8d, 5
  mov r9d, 6
  mov qword ptr [rsp], 7      ; Argument 7
  mov qword ptr [rsp + 8], 8 ; Argument 8
  call qword ptr [rip + example::doit@GOTPCREL]
  add rsp, 24 ; Cleanup Arguments
  ret
```

NOTICE: There are 24 bytes on the stack, but only 16 bytes of arguments.

CALLEE EXAMPLE

```
example::doit:
  sub rsp, 104
  mov rax, qword ptr [rsp + 120]
  mov r10, qword ptr [rsp + 112]
  add rdi, rsi
  seto r11b     ; Overflow check
  test r11b, 1
  jne .LBB0_8   ; Panic on overflow
```

- 104 bytes of stack space
- rsp+120-104 = rsp+16 = arg1
- return address at **rsp**

CLOSURES AND PREDICATES

- Rust has several functional features
- Template generalizations

Save/Load Add new... Vim Rust rustc 1.40.0 Compiler options...

```rust
pub fn main() {
    let xs = [1,2,3,4];
    let sum: u64 =  xs.iter().map(|x| x * 2 as u64).sum();
}
```

```asm
example::main:
  sub rsp, 72
  mov qword ptr [rsp + 32], 1
  mov qword ptr [rsp + 40], 2
  mov qword ptr [rsp + 48], 3
  mov qword ptr [rsp + 56], 4
  lea rax, [rsp + 32]
  mov rdi, rax
  mov esi, 4
  call qword ptr [rip + core::slice::<impl [T]>::iter@GOTPCREL]
  mov qword ptr [rsp + 24], rax
  mov qword ptr [rsp + 16], rdx
  mov rdi, qword ptr [rsp + 24]
  mov rsi, qword ptr [rsp + 16]
  call qword ptr [rip + core::iter::traits::iterator::Iterator::map@GOTPCREL]
  mov qword ptr [rsp + 8], rax
  mov qword ptr [rsp], rdx
  mov rdi, qword ptr [rsp + 8]
  mov rsi, qword ptr [rsp]
  call qword ptr [rip + core::iter::traits::iterator::Iterator::sum@GOTPCREL]
  add rsp, 72
  ret

example::main::{{closure}}:
  sub rsp, 24
  mov qword ptr [rsp + 16], rdi
  mov rdi, rsi
  mov esi, 2
  call <&u64 as core::ops::arith::Mul<u64>>::mul
  mov qword ptr [rsp + 8], rax
  mov rax, qword ptr [rsp + 8]
  add rsp, 24
  ret
```

CHALLENGES

http://ctf.segfault.me/

Have Fun!

HINTS

HINTS

- Flag format is **`FLAG-[a-zA-Z0-9]{37}`**

HINTS

- Flag format is **`FLAG-[a-zA-Z0-9]{37}`**
- Try clearing the blob of strings data definition
  - Clearer XREFs

HINTS

- Flag format is **`FLAG-[a-zA-Z0-9]{37}`**
- Try clearing the blob of strings data definition
  - Clearer XREFs
- Find the **`main`** and **`validate`** symbols

READER HINTS

READER HINTS

- There are 4 parts in the validation routine

READER HINTS

- There are 4 parts in the validation routine
- Each part gates the next parts

READER HINTS

- There are 4 parts in the validation routine
- Each part gates the next parts
- **Static**: Start from the end of the function and work backwards

REVEAL HINTS

REVEAL HINTS

- You need to create a file

REVEAL HINTS

- You need to create a file
- The file must contain a 4 byte key and the flag

REVEAL HINTS

- You need to create a file
- The file must contain a 4 byte key and the flag
- Look for the base64 encoding

REVEAL HINTS

- You need to create a file
- The file must contain a 4 byte key and the flag
- Look for the base64 encoding
- Known Plaintext: **`FLAG-`** should be enough to recover the key

REACTIVATE HINT

- Find where the key is computed and grab it from memory

NOTE: It's probably easier to do this one dynamically

REACTIVATE HINT

- The JSON payload is `{"user":"...","key":"..."}`

- Find where the key is computed and grab it from memory

NOTE: It's probably easier to do this one dynamically

REACTIVATE HINT

- The JSON payload is `{"user":"...","key":"..."}`
- Post to `/activate`

- Find where the key is computed and grab it from memory

NOTE: It's probably easier to do this one dynamically

REACTIVATE HINT

- The JSON payload is `{"user":"...","key":"..."}`
- Post to `/activate`
- Only one valid key per user
- Find where the key is computed and grab it from memory

NOTE: It's probably easier to do this one dynamically

REFERENCES

- https://www.rust-lang.org/
- https://godbolt.org/
- https://rust-lang.github.io/rustc-guide/