ONE THOUSAND WAYS TO DIE IN RUST FFI

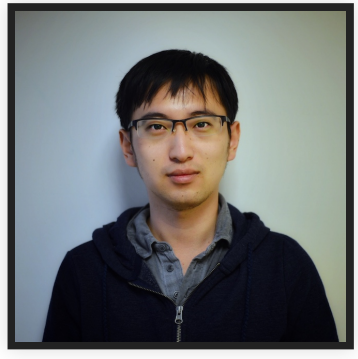
(AND HOW TO SURVIVE)

Yiming Jing

百度安全 有AI更安全 Baidu X-Lab

ABOUT ME

- Security scientist at Baidu X-Lab
- Author of *MesaLink*, providing OpenSSL-compatible C APIs for *rustls*



MesaLink

Daniel Stenberg
@bagder

Follow

With MesaLink, curl now supports 12(!) different TLS libraries...

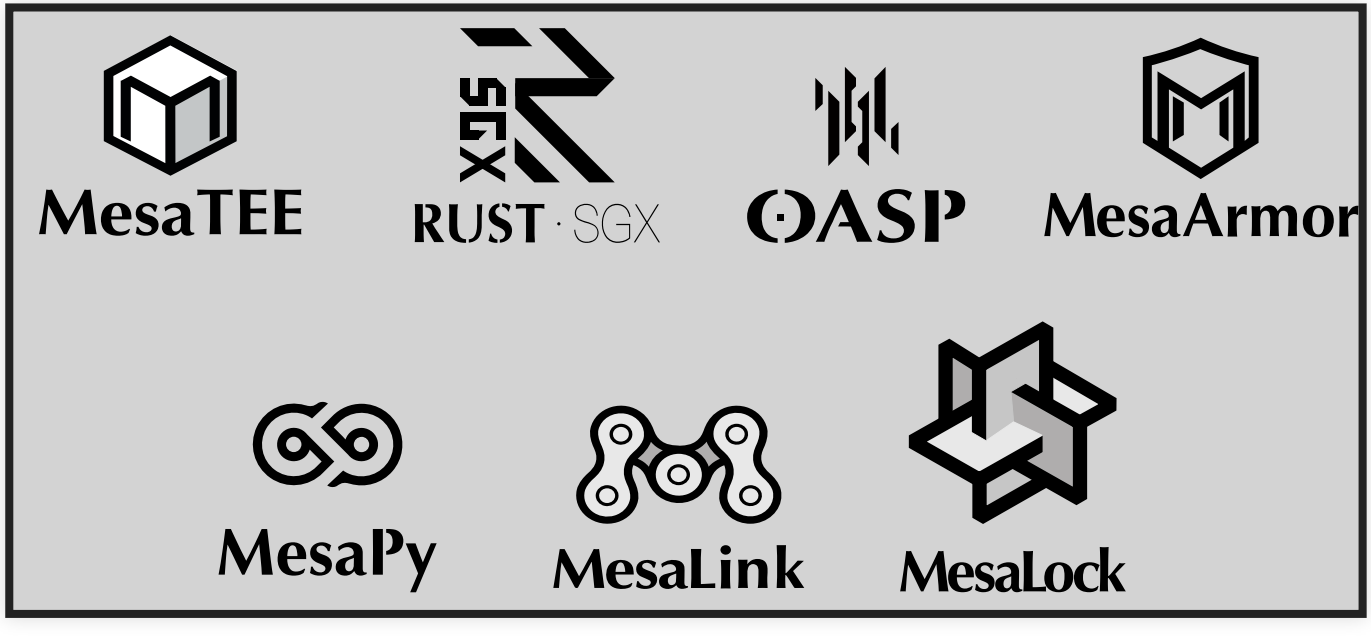
curl commits @curlcommits
github.com/curl/curl/comm… configure.ac: add a MesaLink vtls backend

11:31 PM - 12 Sep 2018

ABOUT BAIDU X-LAB

- Baidu is the 2nd largest search engine in the world
- X-Lab is led by Dr. Lenx Wei, the Chief Security Scientist of Baidu

Memory safety projects lineup



OUTLINE OF THIS TALK

A hello world ... and how it ends in a disaster

10 tips for writing Rust FFI code

Recap and Future Work

✅ for good patterns, ❌ for anti-patterns
⚠️ for "use with caution"

HELLO, RUST FFI

```
// ✅
#[no_mangle]
pub extern fn hello_rust() -> *const u8 {
    "Hello, Rust\0".as_ptr()
}
```

HELLO, RUST FFI

```
1 use libc::c_char;
2 use std::ffi::CStr;
3 // ❌❌❌❌❌
4 #[no_mangle]
5 pub extern "C" fn say_hello_to(who_ptr: *const c_char)
6         -> *const c_char
7     let who_str = unsafe {
8     CStr::from_ptr(who_ptr).to_str().unwrap()
9     };
10
11    format!("Hello, {}", who_str).as_ptr()
12 }
```

who_ptr, unwrap(), format!, as_ptr(), retval

TIP #1: POINTER SANITY CHECKS

Always check for null pointers

```
// ✅
if who_ptr.is_null() {
        return Err(...)
}
```

TIP #1: POINTER SANITY CHECKS

- Check if the dereferenced object is what you want
- Use magic bytes as type identifiers
- Check the bytes immediately after dereferencing

```
// ✅
struct SSL_CTX { magic: u32, ... }
trait OpaquePointerGuard { fn check_magic() -> bool; }
impl OpaquePointerGuard for SSL_CTX {
        fn check_magic(&self) -> bool {
                self.magic == 0xBAAD_CAFE; // can be any value
        }
}
fn SSL_read(ssl_ctx_ptr: *mut SSL_CTX, ....) -> c_int {
        // ... other pointer sanity checks
        let ssl_ctx = unsafe { &mut *ssl_ctx_ptr };
        if !ssl_ctx.check_magic() { return Err(...) }
        ...
```

TIP #2: TYPE CONVERSION

- C strings are nul-terminated arrays of bytes
- Rust strings are UTF-8 encoded bytes

```
// ⚠️
// additional checks needed here
let rust_str = match CStr::from_ptr(c_str_ptr)
    .to_str() {
    Ok(s) => ... ,
    Err(utf8_error) =>  ... ,
};
```

TIP #2: TYPE CONVERSION

- `libc:c_char` is `i8` or `u8`, depending on the target

```
fn ptr_to_u8_slice(ptr: *const libc::c_char) {
        ...
        let slice: &[u8] = unsafe {
                slice::from_raw_parts(ptr)
        }; // ❌: type mismtach
        ...
}
```

- Transmuting `&[c_char]` to `&[u8]`

```
// ✅: https://doc.rust-lang.org/std/mem/fn.transmute.html
let u8_slice = unsafe {
        &*( &slice as *const [c_char] as *const [u8])
};
```

TIP #3: GETTING C POINTERS

- Use as_ptr() for &‘static [u8] literals

```
// ✅
let s: &'static [u8] = b"hello world\0";
let s_ptr = s.as_ptr() as *const c_char;
```

- Create a C-compatible copy with CString

```
// ✅
let c_string = CString::new("hello").unwrap();
let c_string_ptr = c_string.into_raw();
```

TIP #3: GETTING C POINTERS

- Avoid dangling poiners

```
struct Foo(u8);
fn dangling_ptr() -> *const Foo {
        let foo = Foo(1u8);
        let foo_ptr = &foo as *const Foo;
        foo_ptr // ❌: foo does not outlive foo_ptr
}
```

- Transfer ownership to C with `Box::into_raw()`

```
// ✅
struct Foo(u8);
let foo = Box::new(Foo(2u8));
return Box::into_raw(foo);
```

TIP #4: DO NOT ABUSE RAW POINTERS

- Do not cheat the borrow checker with raw point

```
// https://github.com/actix/actix-web/issues/289
struct Foo(String);
let foo = Foo("Some immutable data".into());
let foo_alias: &mut Foo = unsafe {
    &mut *( &foo as *const Foo as *mut Foo )
};
foo_alias.0 = "You don't catch me, borrow checker!!!".into();
```

TIP #5: MEMORY ALLOCATION AND DEALLOCATION

- String, CString, to_string()/into(), format!
- Vec and vec!
- Box, Rc, Arc
- std::collections
- impl Drop
- mem::forget
- Out of scope variables
- into_raw

TIP #6: PANICS

- Unwinding past the FFI boundaries is UB
- Use `catch_unwind` at the boundaries

```
#[no_mangle]
pub extern "C" fn foo() -> c_int {
    // ✅
    match catch_unwind(AssertUnwindSafe(|| {
        ...
    })) {
        Ok(ret) => return ret,
        Err(_) => Your function panicked!!!
     }
}
```

TIP #6: PANICS

- Watch for things that panic at runtime
- `unwrap()`
- `cell::RefCell`
- `slice::copy_from_slice`
- assert!, unimplemented!, unreachable!
- Overflow checks
- Third-party functions: callback and dependencies

TIP #7: CONVERTING FILE DESCRIPTORS

- c_int: use FromRawRd and AsRawFd (Unix only)
- Validate a fd with `libc::fcntl(fd, F_GETFD)`

```
// ✅
use std::os::unix::io::FromRawFd;
let tcp_sock = unsafe {
    TcpStream::from_raw_fd(sock_fd)
};
```

- libc::FILE: use fdopen and fileno

```
// ✅
use libc::{FILE, fdopen};
let f: *mut FILE = unsafe { fdopen(fd) };
```

TIP #8: COPYING DATA FROM C INTO RUST

- `libc::memcpy`, `libc::memmove`
- `ptr::copy_nonoverlapping`
- `ptr::from_raw_parts` + `slice::copy_from_slice`

TIP #9: CONVERTING C ENUMS

```
// ✅ original_compression.h
enum compression_level_t {
    LEVEL_DEFAULT = 2, LEVEL_LOW = 1, LEVEL_HIGH = 2
}
```

But, Rust enums cannot contain duplicate values

```
// ✅ compression.rs
#[repr(u32)]
enum CompressionLevel {
    LevelLow = 1, LevelHigh = 2
}
```

```
// ✅ compression.h
enum compression_level_t {
    LEVEL_LOW = 1, LEVEL_HIGH = 2
}
#define LEVEL_DEFAULT LEVEL_HIGH
```

TIP #9: CONVERTING C ENUMS

Validate inputs from C with the `From`/`Into` trait

```
// ✅
#[repr(u32)]
enum CompressionLevel {
    LevelLow = 1, LevelHigh = 2, LevelUndefined = 0xffff
}
impl From<c_int> for CompressionLevel {
    fn from(val: c_int) -> CompressionLevel {
        match val {
            1 => CompressionLevel::LevelLow,
            2 => CompressionLevel::LevelHigh,
            _ => CompressionLevel::LevelUndefined,
        }
    }
}
```

TIP #9: CONVERTING C ENUMS

`#[derive(Debug)]` and `*char` point

```
fn get_compression_level_name_ptr(level: CompressionLevel)
    -> *const c_char {
    format!("{:?}", level).as_ptr() as *const c_char
    // ❌: Incorrect! See Tip #2, #5
}

use std::ffi::CString;
fn get_compression_level_name_ptr(level: CompressionLevel)
    -> *const c_char {
    CString::new(format!("{:?}", level))
        .unwrap()
        .into_raw() as *const c_char
    // ❌: It works but allocates memory. See Tip #5, #6
}
```

✅ Use our crate: `enum_to_u8_slice_derive`

TIP #10: EXTERNAL TOOLS

- Bindgen, CBindgen, safe_bindgen
- ffi_helpers, easy_ffi
- cargo clippy
- Valgrind memcheck
- -Z sanitizer
- cargo fuzz

ACKNOWLEDGEMENTS

- The Rustonomicon
- The unofficial FFI book, by @Michael-F-Bryan
- The Rust FFI Omnibus, by @shepmaster
- rust-ffi-examples, by @alexcrichton
- Previous talks on RustConf and RustFest

RECAP

1. POINTER SANITY CHECKS

2. TYPE CONVERSION BETWEEN C AND RUST TYPES

3. GETTING C POINTERS FROM RUST OBJECTS

4. RAW POINTERS ARE NOT FOR BYPASSING THE BORROW CHECKER

5. MEMORY ALLOCATION AND DEALLOCATION

6. PANICS AND UNWINDING

7. CONVERTING FILE DESCRIPTORS

8. COPYING DATA FROM C INTO RUST

9. CONVERTING C ENUMS INTO RUST

10. EXTERNAL TOOLS

THANKS

🔗 https://mesalink.io

✉️ jingyiming@baidu.com

百度安全 有AI更安全 Baidu X-Lab

Follow us on Twitter: @BaiduXlab