RustZone: Writing Trusted Applications in Rust

Eric Evenchick
Black Hat Europe 2018
About Me

• Principal Research Consultant @ Atredis Partners

• Founder, Developer of Open Source Hardware Things @ Linklayer Labs
Outline

• Trusted Execution Environments
• TrustZone
• TEE Problems
• Rust
• Rust + TrustZone
• Demo
• Questions
Trusted Execution Environments
What?

- An isolated environment within a processor for performing secure operations
- Segmentation of code, data, and hardware access
- Combination of hardware features and software
Today’s TEEs

• Hardware:
  • AMD: Platform Security Processor
  • Intel: Trusted Execution Technology, Software Guard Extensions (SGX)
  • ARM: TrustZone

• Software:
  • Trustonic Kinibi
  • Qualcomm QSEE
  • OP-TEE
Use Cases

- Authentication
  - Android GateKeeper
- Financial Applications
- Secure Boot
- DRM
  - WideVine
- An additional layer of protection from the host OS
- Protect the system from the user 😞
TrustZone
The TrustZone TEE

• The ARM TEE
• Normal and Secure *Worlds*
• Normal World: Rich OS and applications (Linux, Android, QNX, etc...)
• Secure World: Limited operating system and Trusted Applications
• Processor can switch between two worlds
• Configure processor to restrict access to resources
TrustZone in Practice

http://genode.org/documentation/articles/trustzone
TEE Problems
TEE OS Protections

• ASLR is Rare
• No Stack Canaries or Guard Pages
• Secure World has fewer protections than Normal World?
• No High Level Language Support, we must write C!
Writing (good) C is Hard

• Common Memory Problems
  • Buffer overflows
  • Use after free

• Type Issues
  • Void means nothing, and everything!

• Limited Help from Compiler

• Programmers can do Silly Things
  • memcpy, strcpy, sprintf, etc...
Example: WideVine Trusted Application

• DRM Implementation for Android
• Undocumented Command with Buffer Overflow
• End Result: Arbitrary Code Execution in Secure World

• More info: http://bits-please.blogspot.ca/2016/05/qsee-privilege-escalation-vulnerability.html
Example: Samsung OTP Buffer Overflow

- Service in Normal World to generate a One-Time Password (OTP)
- Any user can access this service!
- Trusted Application parses request leading to stack buffer overflow
Rust
What’s Rust?

• New systems programming language
• In development since 2010, sponsored by Mozilla
• Works for embedded:
  • Works without libc
  • Compiles to bytecode
  • No garbage collection or runtime
  • Raw memory access
Why Rust?

• Compile time memory safety checks
• Memory ownership and borrow checking
• Find bugs at compile time, not runtime
  • eg, match
• Good tools, getting better
• Great C Foreign Function Interface!
Rust / C FFI

• Call C from Rust and Call Rust from C

• Need unsafe blocks for:
  1. Dereferencing a raw pointer
  2. Calling an unsafe function or method
  3. Accessing or modifying a mutable static variable
  4. Implementing an unsafe trait

• Goal: limit unsafe code
Learning Rust

• The Rust Book: https://doc.rust-lang.org/book/
  • Paper version soon: https://nostarch.com/Rust
• Rust by Example: https://rustbyexample.com/
• Julia Evans’ Blog: https://jvns.ca/categories/rust/
Rust + TrustZone
Step 1: Get an OS

• Need an OS to run in the Secure World
• OP-TEE
  • Free and Open Source
  • Implementations for many platforms, including QEMU
  • Well Documented
  • [https://www.op-tee.org/](https://www.op-tee.org/)
Step 2: Generate Rust Bindings

• We need Rust bindings for OP-TEE’s API
• bindgen to the rescue!

```rust
void TEE_MACInit(
    TEE_OperationHandle operation, const void *IV,
    uint32_t IVLen);

extern "C" {
    pub fn TEE_MACInit(operation: TEE_OperationHandle,
                        IV: *const c_types::c_void, IVLen: u32);
}
```
Step 3: Write a Rust Library

• Yes, a *library*.

• Need to implement 5 functions:
  • TA_CreateEntryPoint
  • TA_DestroyEntryPoint
  • TA_OpenSessionEntryPoint
  • TA_CloseSessionEntryPoint
  • TA_InvokeCommandEntryPoint
Step 3: Write a Rust Library

```rust
pub fn InvokeCommandEntryPoint(_sessionContext: *mut c_types::c_void,
                             commandID: u32, _paramTypes: u32,
                             params: &mut [optee::TEE_Param; 4]) ->

optee::TEE_Result
{
    ta_print!("Rust TA InvokeCommandEntryPoint");
    match commandID {
        0 => {
            unsafe {params[0].value.a += 1};
            ta_print!("Incremented Value");
        },
        1 => {
            unsafe {params[0].value.a -= 1};
            ta_print!("Decremented Value");
        },
        _ => {
            return optee::TEE_ERROR_BAD_PARAMETERS;
        }
    }
    return optee::TEE_SUCCESS;
}
```
Step 4: Compile, Link, Sign

- Compiled Rust Library
- Compiled TA Header
- libutee, libmpa, libutil
- OP-TEE Linker Script

Linker → TA ELF → sign.py → Signed TA
Demo
Conclusions
Conclusions

• TEEs are useful, but have the usual issues
• Rust is an potential replacement for C with some added benefits
• Should you write your Trusted Applications in Rust?
Thanks! Questions?

eric@evenchick.com
@ericevenchick
https://github.com/ericevenchick/rustzone