Bringing Memory-Safety to Keystone Enclave

Mingshen Sun
Baidu X-Lab
Open-Source Enclaves Workshop (OSEW 2019)
Berkeley, July 2019
A Framework for Universal Secure Computing

Highlights

- MesaTEE is the next-gen solution to enable general computing service for security-critical scenarios. It will allow even the most sensitive data to be securely processed to enable offshore businesses without leakage.

- The solution combines the advanced Hybrid Memory Safety (HMS) model and the power of the Trusted Computing technologies (e.g. TPM) as well as the Confidential Computing technologies (e.g. Intel® SGX).

What Is MesaTEE?

The emerging technologies of big data analytics, machine learning, cloud/edge computing, and blockchain are significantly boosting our productivity, but at the same time they are bringing new confidentiality and integrity concerns. On public cloud and blockchain, sensitive data like health and financial records may be consumed at runtime by untrusted computing processes running on compromised platforms; during inhouse data exchange, confidential information may leak. "Big Data" and "Blockchain" technologies, although extremely innovative, need to be secured in order to provide any meaningful privacy issue.

Although the security of the implementation of the secondary computing algorithms themselves, such as AI models, also need to be well protected. Once leaked, attackers can steal the intellectual properties, or launch whitebox attacks and easily exploit the weakness of the models.

Facing all these risky scenarios, we are in desperate need of a trusted and secure mechanism, enabling us to protect both private data and proprietary computing models during a migratable execution in potentially unsafe environments, yet preserving functionalities, performance, compatibility, and flexibility. MesaTEE is targeting to be, as we call it, the full "Universal Secure Computing" stack, so it can help users resolve these runtime security risks.

https://mesatee.org
Rust and Keystone Enclave

- **Open-Source Enclave (RISC-V Hardware/Keystone Enclave):** openness, simplicity, and flexibility
- **Rust:** Safety, performance, and productivity

**Outline:**

1. Briefly introduce our progress in bringing Rust to Keystone Enclave
2. Discuss our efforts in implementing safe GlobalPlatform TEE APIs implemented in OP-TEE
Why Rust

• **Memory-safety issues** break security guarantees of TrustZone.


Rust and Keystone Enclave

- Host libraries (lib/host)
- Enclave Application libraries (lib/app)
- Edge libraries (lib/edge)
- Runtimes (rts/)
Rust Keystone

- Target: riscv64imac-unknown-none-elf

```rust
26 #![link_section=".text._start"]
27 #![no_mangle]
28 extern "C" fn eapp_entry() {
29     let msg: *const u8 = b"Hello, World!
\0".as_ptr();
30     let ret: u64 = 42;
31     unsafe {
32         ocall(OCALL_PRINT_VALUE, &ret as *const u64 as u64, 8, 0, 0);
33         ocall(OCALL_PRINT_BUFFER, msg as u64, 15, 0, 0);
34     }
35     let mut state = State::<ARC>::new();
36     state.update(b"12345");
37     state.update(b"6789");
38     unsafe {
39         let crc: u64 = state.get() as u64;
40         ocall(OCALL_PRINT_VALUE, &crc as *const u64 as u64, 8, 0, 0);
41         let fib_ret = fib(35);
42         ocall(OCALL_PRINT_VALUE, &fib_ret as *const u64 as u64, 8, 0, 0);
43         EAPP_RETURN(fib_ret);
44     }
45 }
```
TEE Specs

- **GlobalPlatform** TEE specifications
  - *TEE System Architecture (GPD_SPE_009)*: defines a general TEE architecture
  - *TEE Internal Core API Specification (GPD_SPE_010)*
  - *TEE Client API Specification (GPD_SPE_007)*: defines communication interface between Rich OS apps and trusted apps.

- **OP-TEE**: open portable trusted execution environment in compliance with GlobalPlatform specs.
TrustZone Architecture

- **Normal World**
  - EL0: client apps
  - EL1: Rich OS
  - EL2: Hypervisor

- **Secure World**
  - EL0: trusted apps
  - EL1: Trusted OS
  - EL2: Secure Monitor

- **Secure Worlds**
  - S-EL0
  - S-EL1
  - S-EL3
OP-TEE Implementation

- **Normal World**
  - client apps
  - TEE Client SDK
  - TEE Supplicant
  - Rich OS (OP-TEE driver)

- **Secure World**
  - trusted apps
  - TEE TA SDK
  - OP-TEE Trusted OS

- **ARM Trusted Firmware (Secure Monitor)**
Safe SDK Design

**Normal World**
- client apps
- TEE Client SDK
- Rich OS (OP-TEE driver)

**Secure World**
- trusted apps
- TEE TA SDK
- OP-TEE Trusted OS

**Safe GlobalPlatform**
- TEE Client API
- TEE Internal Core API

**Rust OP-TEE**
- TrustZone SDK

**ARM Trusted Firmware (Secure Monitor)**
Design of Client SDK

Client apps targets:
- aarch64-unknown-linux-gnu
- arm-unknown-linux-gnu
Design of TA SDK

Two new targets in the Rust compiler/std:
- aarch64-unknown-optee-trustzone
- arm-unknown-optee-trustzone
Project Structure

- `optee-teec`: client-side Rust library (LoC: ~933)
- `optee-utee`: TA-side Rust library (LoC: ~2827)
- `optee`: upstream optee library (`optee_client`, `optee_os`)
- `rust`: modified Rust including
  - `rust`: ~29 files changed, 1800 insertions
  - `libc`: ~4 files changed, 131 insertions
  - `compiler-builtins`: ~3 files changed, 3 insertions(+) , 1 deletion(-)
- `examples`: `hello_world`, `aes`, `hotp`, `random`, `secure_storage`, and `serde` (LoC: ~3373)
Example - Demo in QEMU
GlobalPlatform TEE API Specification

Normal World

- TEEC_InitializeContext
- TEEC_OpenSession
- TEEC_InvokeCommand
- TEEC_CloseSession

Secure World

- TA_CreateEntryPoint
- TA_OpenSessionEntryPoint
- TA_InvokeCommandEntryPoint
- TA_CloseSessionEntryPoint
- TA_DestroyEntryPoint

①
②
③
④
Example - Client (Current Design)

```rust
code
use optee_tee::{Context, Operation, ParamType, Session, Uuid};
use optee_tee::{ParamNone, ParamValue};
use proto::{self, Command};

fn hello_world(session: &mut Session) -> optee_tee::Result<()> {  
  let p0 = ParamValue::new(29, 0, ParamType::ValueInout);
  let mut operation = Operation::new(0, p0, ParamNone, ParamNone, ParamNone);

  println!("original value is {:#?}", operation.parameters().0.a());

  session.invoke_command(Command::IncValue as u32, &mut operation)?;
  println!("inc value is {:#?}", operation.parameters().0.a());

  session.invoke_command(Command::DecValue as u32, &mut operation)?;
  println!("dec value is {:#?}", operation.parameters().0.a());
  Ok(())
}

fn main() -> optee_tee::Result<()> {  
  let mut ctx = Context::new()?;
  let uuid = Uuid::parse_str(proto::UUID).unwrap();
  let mut session = ctx.open_session(uuid)?;

  hello_world(&mut session)?;

  println!("Success");
  Ok(())
}
```
Example - Trusted App (First Commit)

```rust
#![allow(non_upper_case_globals)]
#![allow(non_camel_case_types)]
#![allow(non_snake_case)]

extern crate optee_utee;
pub use optee_utee::*;

#[no_mangle]
pub extern "C" fn TA_CreateEntryPoint() -> TEE_Result {
    return TEE_SUCCESS;
}

#[no_mangle]
pub extern "C" fn TA_OpenSessionEntryPoint(_paramTypes: ParamTypes, _params: TEE_Param, _sessionContext: SessionP) -> TEE_Result {
    return TEE_SUCCESS;
}

#[no_mangle]
pub extern "C" fn TA_DestroyEntryPoint() {
}

#[no_mangle]
pub extern "C" fn TA_OpenSessionEntryPointPoint(_sessionContext: SessionP) {
}

#[no_mangle]
pub extern "C" fn TA_CloseSessionEntryPoint(_sessionContext: SessionP) {
}

#[no_mangle]
pub extern "C" fn TA_InvokeCommandEntryPoint(_sessionContext: SessionP, _command: u16, _paramCount: u16, _params: [u8; 32]) -> TEE_Result {
    match _commandID {
        0 => {
            unsafe { _params[0].value.a += 121; }
        },
        1 => {
            unsafe { _params[0].value.a -= 21; }
        },
        _ => {
            return TEE_ERROR_BAD_PARAMETERS;
        }
    }
    return TEE_SUCCESS;
}
```
Example - Trusted App (Current Design)

```rust
#[ta_create]
fn create() -> Result<()> {
    trace_println("[*] TA create");
    Ok(())
}

#[ta_open_session]
fn open_session(params: &mut Parameters) -> Result<()> {
    trace_println("[*] TA open session");
    Ok(())
}

#[ta_close_session]
fn close_session() {
    trace_println("[*] TA close session");
}

#[ta_destroy]
fn destroy() {
    trace_println("[*] TA destroy");
}

#[ta_invoke_command]
fn invoke_command(cmd_id: u32, params: &mut Parameters) -> Result<()> {
    trace_println("[*] TA invoke command");
    let mut values = unsafe { params.0.as_value().unwrap() };
    match Command::from(cmd_id) {
        Command::IncValue => {
            values.set_a(values.a() + 100);
            Ok(())
        }
        Command::DecValue => {
            values.set_a(values.a() - 100);
            Ok(())
        }
        _ => Err(Error::new(ErrorKind::BadParameters)),
    }
}
```
Example - Use Serde

```rust
#[ta_invoke_command]
fn invoke_command(cmd_id: u32, _params: &mut Parameters) -> Result<()> { 
    trace_println!("[+] TA invoke command"); 
    match Command::from(cmd_id) { 
        Command::DefaultOp => { 
            let point = Point { x: 1, y: 2 }; 

            // Convert the Point to a JSON string. 
            let serialized = serde_json::to_string(&point).unwrap(); 

            // Prints serialized = {"x":1,"y":2} 
            trace_println!("serialized = {}", serialized); 

            // Convert the JSON string back to a Point. 
            let deserialized: Point = serde_json::from_str(&serialized).unwrap(); 

            // Prints deserialized = Point { x: 1, y: 2 } 
            trace_println!("deserialized = {?}" , deserialized); 
            Ok(())
        } 
        _ => Err(Error::new(ErrorKind::BadParameters)), 
    }
}
```
Other Examples

• **hello_world**: minimal project structure

• **aes**: crypto, shared memory APIs

• **hotp**: crypto APIs

• **random**: crypto APIs

• **secure_storage**: secure object related APIs

• **serde**: Rust third-party crates for de/serialization

• **message_passing_interface**
Thanks

- Rust and Keystone Enclave

- Safe GlobalPlatform APIs implemented in OP-TEE

- Baidu ❤️ Rust
  - Rust SGX SDK
  - MesaTEE: A Framework for Universal Secure Computing
  - MesaLock Linux, MesaLink, MesaPy, etc.
Backup Slides
Example - Client (Initial Design)

```cpp
unsafe {
    raw::TEEC_Context
    raw::TEEC_Session
    raw::TEEC_Parameter
    raw::TEEC_Operation
    raw::TEEC_InitializeContext
    raw::TEEC_OpenSession
    raw::TEEC_InvokeCommand
    raw::TEEC_CloseSession
    raw::TEEC_FinalizeContext
}
```
Example - Project Structure

- **host/**: source code of the client app
- **ta/**: source code of TA
  - **ta.lds**: linker script
  - **Xargo.toml**: "Cargo.toml" for cross compilation
  - **ta_static.rs**: some static data structure for TA
- **proto/**: shared data structure and configurations like a protocol
- **Makefile**: Makefile to build host and client
- **uuid.txt**: UUID for TA, randomly generated if the file does not exist.
Project Structure - rust/libstd

src/librustc_target/spec/aarch64_unknown_optee_trustzone.rs

src/libstd/sys/optee/alloc.rs
src/libstd/sys/optee/args.rs
src/libstd/sys/optee/backtrace.rs
src/libstd/sys/optee/cmath.rs
src/libstd/sys/optee/condvar.rs
src/libstd/sys/optee/env.rs
src/libstd/sys/optee/fs.rs
src/libstd/sys/optee/io.rs
src/libstd/sys/optee/memchr.rs
src/libstd/sys/optee/mod.rs
src/libstd/sys/optee/mutex.rs

src/libstd/sys/optee/net.rs
src/libstd/sys/optee/os.rs
src/libstd/sys/optee/os_str.rs
src/libstd/sys/optee/path.rs
src/libstd/sys/optee/pipe.rs
src/libstd/sys/optee/process.rs
src/libstd/sys/optee/rwlock.rs
src/libstd/sys/optee/stack_overflow.rs
src/libstd/sys/optee/stdio.rs
src/libstd/sys/optee/thread.rs
src/libstd/sys/optee/thread_local.rs
src/libstd/sys/optee/time.rs
Example: alloc.rs

```rust
use crate::libc;
use crate::alloc::{GlobalAlloc, Layout, System};

#![stable(feature = "alloc_system_type", since = "1.28.0")]
unsafe impl GlobalAlloc for System {
    #[inline]
    unsafe fn alloc(&self, layout: Layout) -> *mut u8 {
        libc::malloc(layout.size()) as *mut u8
    }

    #[inline]
    unsafe fn alloc_zeroed(&self, layout: Layout) -> *mut u8 {
        libc::calloc(layout.size(), 1) as *mut u8
    }

    #[inline]
    unsafe fn dealloc(&self, ptr: *mut u8, _layout: Layout) {
        libc::free(ptr as *mut libc::c_void)
    }

    #[inline]
    unsafe fn realloc(&self, ptr: *mut u8, _layout: Layout, new_size: usize) -> *mut u8 {
        libc::realloc(ptr as *mut libc::c_void, new_size) as *mut u8
    }
}
```

The underlying library of libc is libutil from OP-TEE
Example: thread.rs

```rust
crate::boxed::FnBox;
crate::ffi::CStr;
crate::io;
crate::sys::{unsupported, Void};
crate::time::Duration;

pub struct Thread(Void);

pub const DEFAULT_MIN_STACK_SIZE: usize = 4096;

impl Thread {
    // unsafe: see thread::Builder::spawn_unchecked for safety requirements
    pub unsafe fn new(_stack: usize, _p: Box<dyn FnBox>>)
        -> io::Result<Thread>
    {
        unsupported()
    }

    pub fn yield_now() {
        panic!("unsupported")
    }

    pub fn set_name(_name: &CStr) {
        panic!("unsupported")
    }

    pub fn sleep(_dur: Duration) {
        panic!("unsupported");
    }

    pub fn join(self) {
        match self.0 {
        }
    }

    pub mod guard {
        pub type Guard = !;
        pub unsafe fn current() -> Option<Guard> { None }
        pub unsafe fn init() -> Option<Guard> { None }
    }
}
Background

- **ARM TrustZone** provide *trusted execution environment* in mobile phone and embedded devices

- TrustZone secures mobile payment, identification authentication, key management, AI models, DRM, OS integrity, etc.
TrustZone Architecture

An Exploration of ARM TrustZone Technology: https://genode.org/documentation/articles/trustzone
Project Structure

- Rust OP-TEE TrustZone SDK: https://github.com/mesalock-linux/rust-optee-trustzone-sdk
- Rust: https://github.com/mesalock-linux/rust
- Rust libc: https://github.com/mesalock-linux/libc.git
- Rust compiler-builtins: https://github.com/mesalock-linux/compiler-builtins.git
Other Examples

- **hello_world**: minimal project structure
- **aes**: crypto, shared memory APIs
- **hotp**: crypto APIs
- **random**: crypto APIs
- **secure_storage**: secure object related APIs
- **serde**: Rust third-party crates for de/serialization
- **message_passing_interface**
Roadmap

• **April**: open source

• **May**: trusted storage API design, cryptographic operations API design, TEE arithmetical API design, and more third-party Rust crates

• **Jun**: push modified Rust compiler/std to upstream and make OP-TEE TrustZone as an official target.

• **2019 Q3/4**: more trusted apps such as secure key service, remote attestation, fTPM, and machine learning algorithm.