

# RusTEE: Developing Memory-Safe ARM TrustZone Applications

Shengye Wan, Mingshen Sun, Kun Sun, Ning Zhang, Xu He



December 10th, 2020

# OUTLINE

- Introduction
- System Overview
- System Evaluation
- Takeaways

---

# INTRODUCTION

---

# Mobile Devices Are Not Safe

- Mobile devices are facing many different threats
  - Trojans, spyware, ransomware
- We need an unified security solution
  - Mobile devices are mostly shipped with ARM-based chips

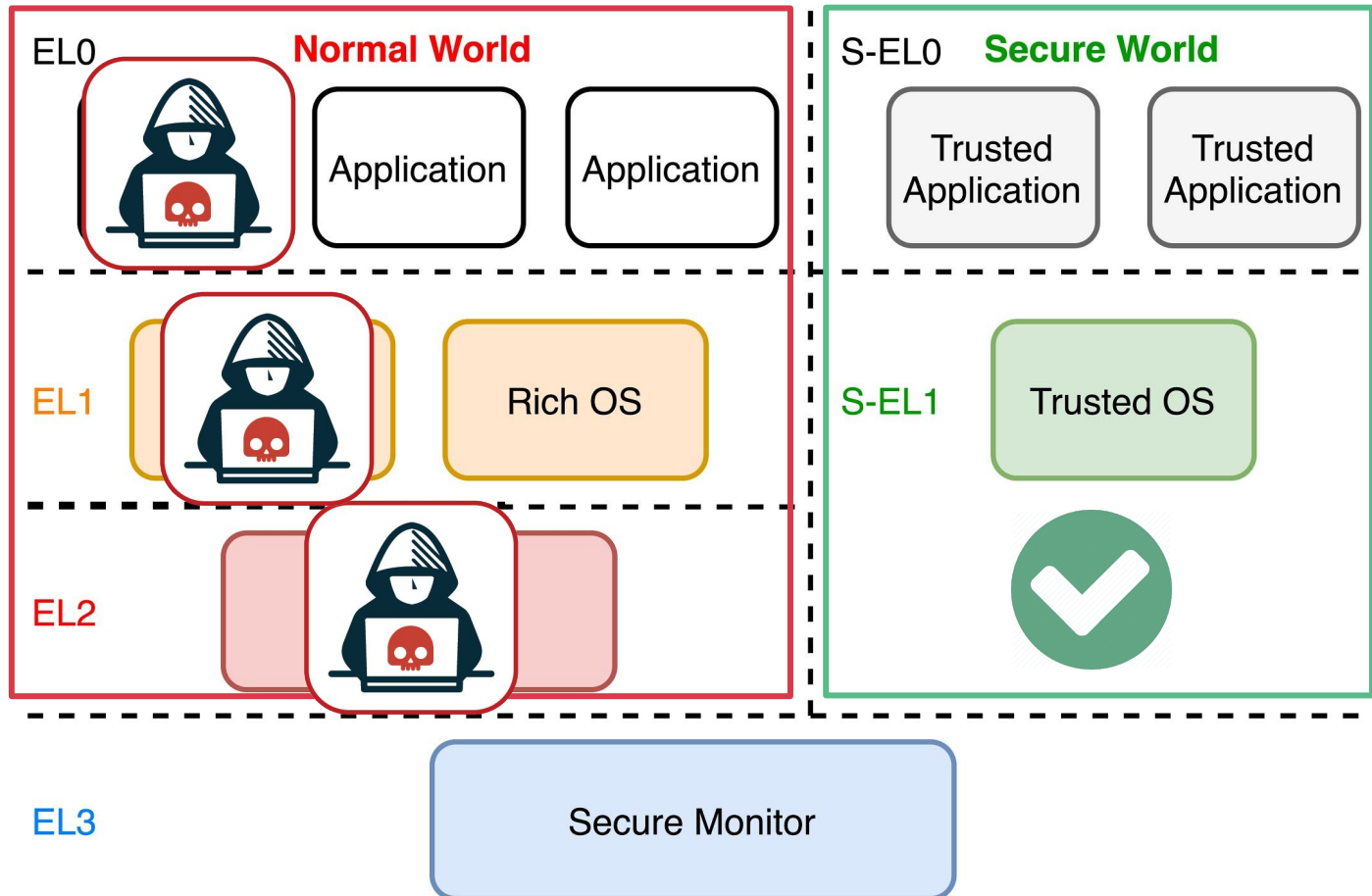


# ARM TrustZone Technology

Entire normal world is **untrusted**  
It is **isolated** from secure components

Secure world is **trusted**

Low  
Privilege



# One TA Breaks Entire Samsung TrustZone

---

```
/* CA passes parameters to TA */
uint32_t v1 = REE_Param[0].val;
uint32_t v2 = REE_Param[1].val;
/* TA uses the values of CA uncarefully */
uint32_t src = v1 + m;
uint32_t length = v2 + n;
/* TA conducts dangerous operation */
memcpy (dest, src, length);
```

**Attacker-controlled parameters**

**Manipulated dangerous behavior**

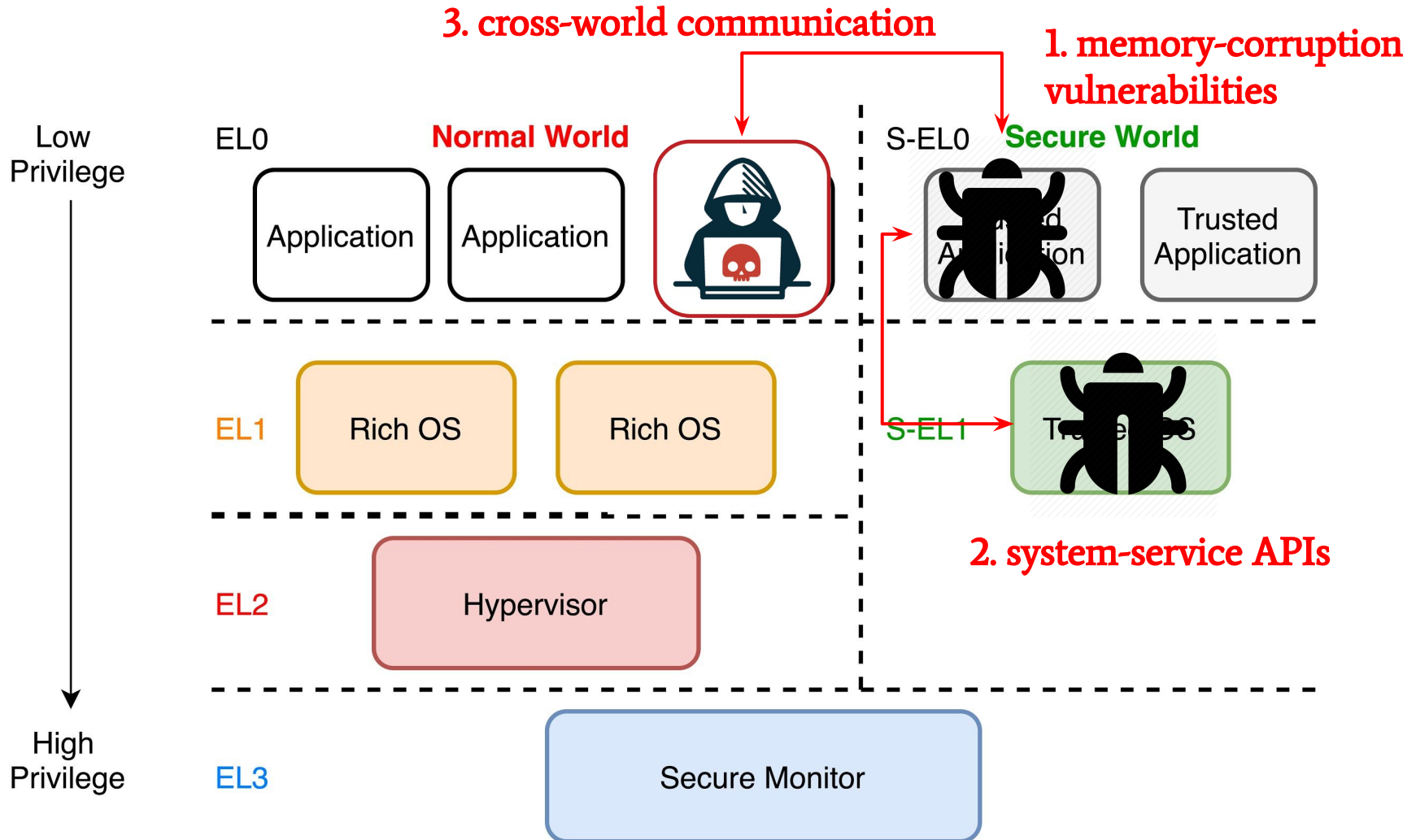
---

C-Style pseudo-code of a vulnerable trusted application,  
CVE-2018-14491 [1]

# Motivation: Trusted Applications Are Vulnerable

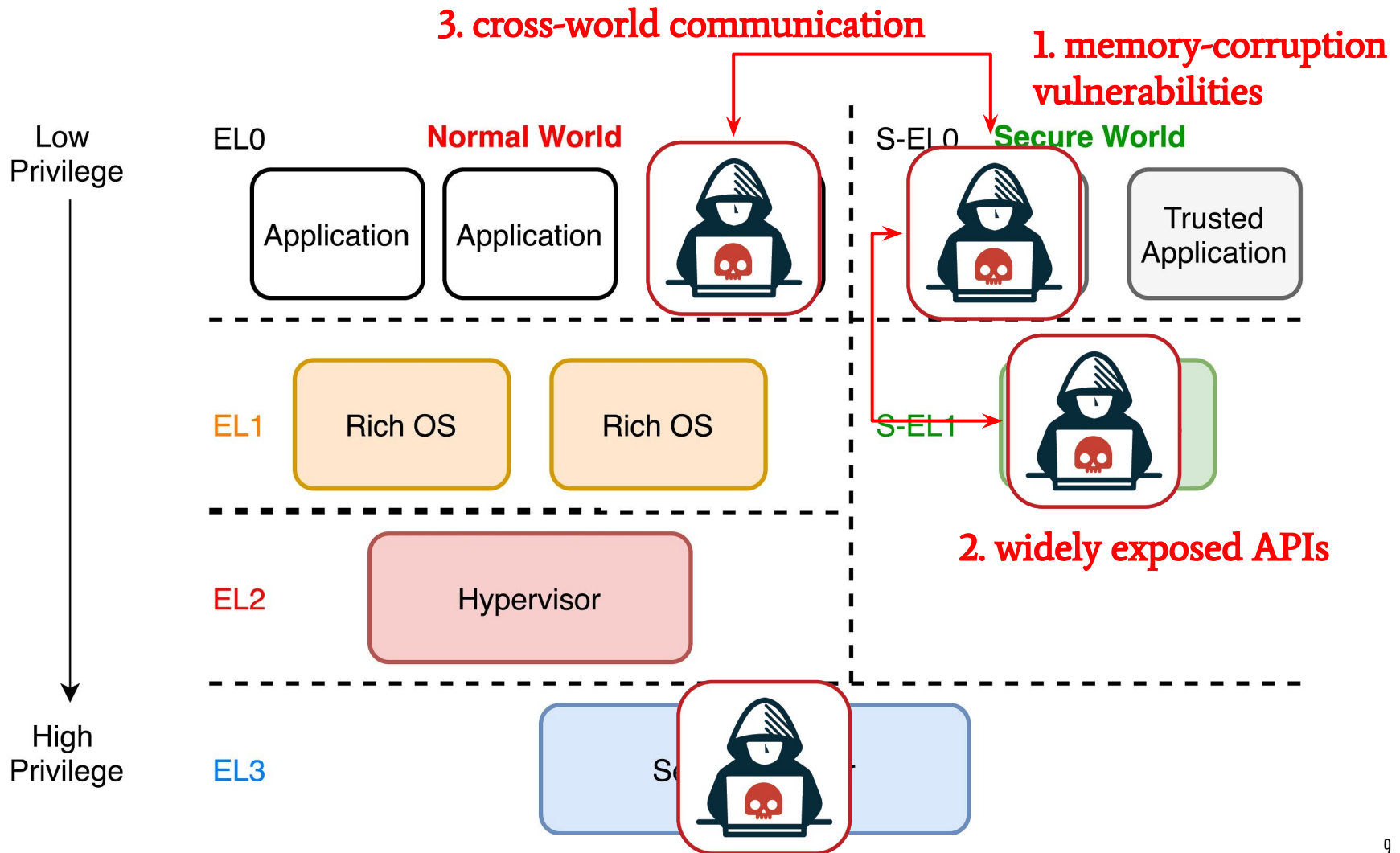
- Issue-1: TAs are written with memory-corruption bugs
  - Memory-unsafe languages: C & assembly code
  - Good performance vs. memory-corruption vulnerabilities
- Issue-2: A vulnerable TA threatens entire secure world
  - Widely exposed system-service APIs
  - user mode -> kernel mode
- Issue-3: A vulnerable TA can get manipulated
  - Cross-world communication
  - Malicious Normal World application can exploit Secure World vulnerabilities

# Security Issues of Trusted Applications





# Security Issues of Trusted Applications



# SYSTEM OVERVIEW

---

# Our Solution: RustEE

- Providing a reliable trusted-application SDK
  - **Key idea:** building trusted applications in the memory-safe language Rust
- Rust language [3]
  - Reliability: promise the memory and thread safety
  - Performance: run-time behavior similar to C
  - Productivity: million crates (libraries)

# Resolving Issue-1: Integrating Rust

- Supporting standard Rust-safe operations
  - Rust compiler can detect memory-corruption bugs for Rust-safe code
  - Manually connecting the trusted OS's standard library with Rust
    - We provide supports for Aarch32 and Aarch64 trusted OS
- Trusted applications still require C-based libraries
  - System services (e.g., cryptography) and cross-world communication
  - Rust imports C libraries via *Foreign Function Interface (unsafe bindings)*
    - Rust compiler skips checking on Rust-unsafe
    - Introducing potential threats

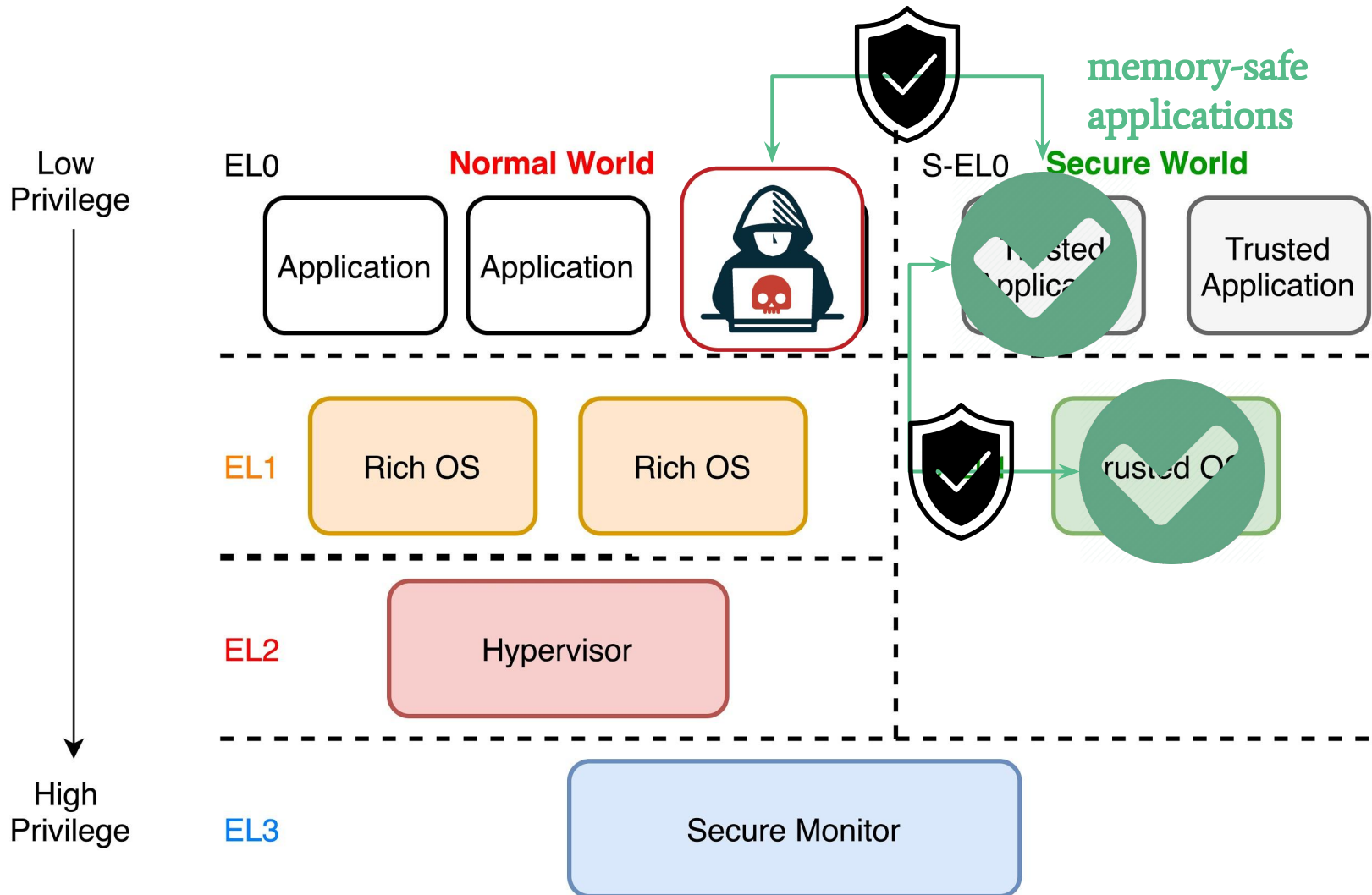
# Resolving Issue-2: Binding Unsafe APIs

- Enforcing 6 principles for binding C-based libraries
  - Adapting 4 principles of Rust-SGX [4]
    - *Bytes, ContiguousMemory, Sanitizable[T], Handle<sub>τ</sub>*
  - Proposing 2 new principles for binding TrustZone-specific APIs
    - i. Enforcing the serialization of grouped APIs
      - API-prepare -> API-encrypt -> API-finalize
    - ii. Enforcing allocation & release for sensitive data structures
      - Example: *impl Drop for OperationHandle {}*

# Resolving Issue-3: Securing Communication

- 4 involved data structures
  - Context, Session, Command, Parameter
- 3 security enhancements
  1. Management of all structures' *lifetimes*
  2. Management of Parameter's *mutability* (R/W permission)
  3. Enforcing the *type-safety* of Parameter

# Resolving Security Issues



# SYSTEM EVALUATION

---



# RusTEE Implementation

- Implementing our prototype based on OP-TEE OS [5]
  - TAs can be developed with all functionalities of OP-TEE
  - Providing normal-world SDK as the complementary component
- Providing 13 examples
  - Cryptography (e.g., AES, HMAC), file storage, big-number calculation, etc.
  - Re-implement all 6 examples of OP-TEE
- Open-source project [6]
  - More than 8000 Lines-of-Code
  - <https://github.com/scommunity/rust-optee-trustzone-sdk>



[5] Linaro. [OPTEE Secure OS](#). GitHub.

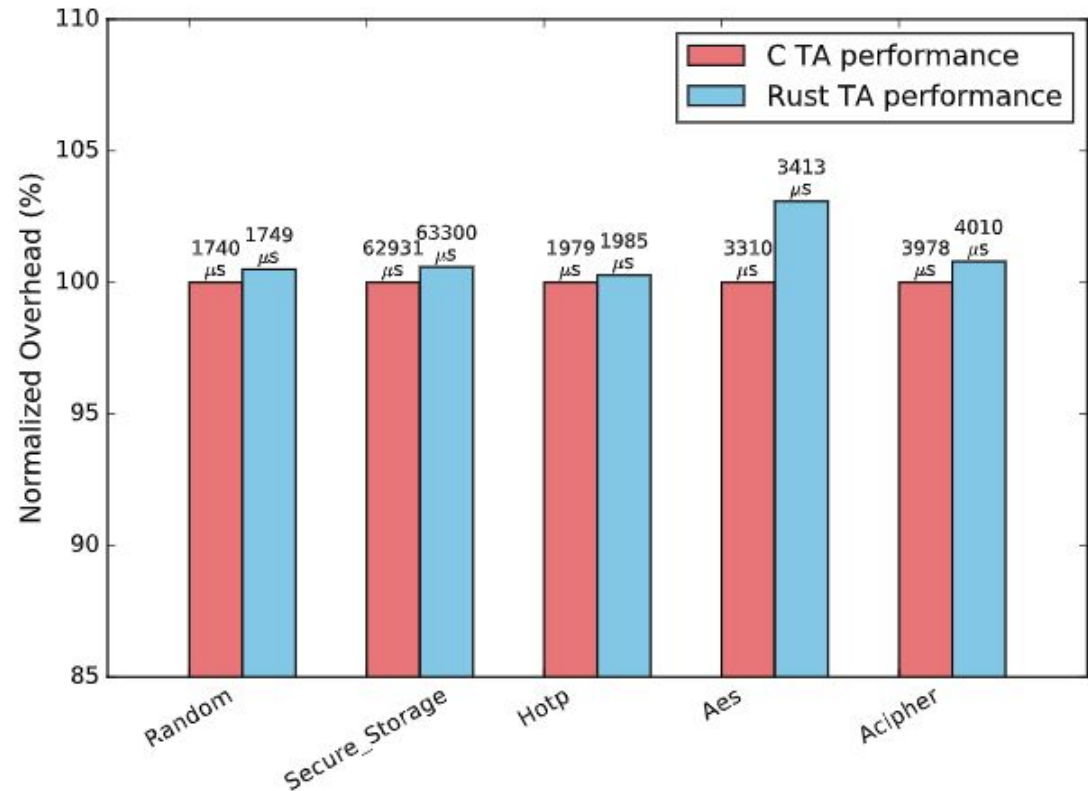
[6] Mesalock Linux. [rust-optee-trustzone-sdk](#). GitHub.

# RusTEE Evaluation

- RusTEE applications vs. OP-TEE applications

- Overhead

- Min = 0.27%
- Max = 3.08%
- Average  $\leq 1\%$




# TAKEAWAYS

---

# Summary

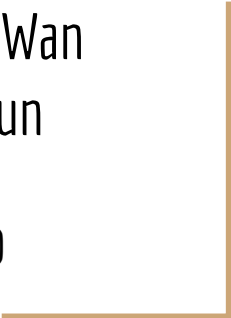
1. We need memory-safe TrustZone Trusted Applications
2. Rust can contribute on building reliable TAs
3. The Trusted OS should interact with TAs carefully
4. TAs should use the data from Normal World carefully



# Thanks & Questions?

Presenter: Shengye Wan  
Q&A: Mingshen Sun

December 10th, 2020



# Motivation: Trusted Applications Are Vulnerable

- Trusted Applications (*TAs*) suffers security issues [2]
  - Implementation Issues
    - Bugs related to specific implementation details
  - Architectural Issues
    - Shared design flaws among different systems

# Mitigating Security Issues of Trusted Applications

- Previous mitigations
  - Isolating the executions of trusted applications
    - Isolating in the normal world [2]
    - Isolating in the time-slice fashion [3]
  - Limitation: introducing non-negligible performance overhead
  
- **Question:** How to build memory-safe trusted applications?

[2] Brasser et al., “SANCTUARY: ARMing TrustZone with User-space Enclaves”

[3] Sun et al., “TrustICE: Hardware-assisted isolated computing environments on mobile devices”