PartEmu: Enabling Dynamic Analysis of Real-World TrustZone Software Using Emulation

Lee Harrison, Hayawardh Vijayakumar, Michael Grace
Samsung Knox, Samsung Research America

Rohan Padhye, Koushik Sen
EECS Department, University of California, Berkeley
The “Hidden” Software Stack: TrustZone
The “Hidden” Software Stack: TrustZone

- Separate software stack
  - Trusted applications (TAs)
  - TrustZone OS (TZOS)
The “Hidden” Software Stack: TrustZone

- Normal World/REE
  - Android apps
  - Android framework
  - OS: Linux kernel
- Secure World/TEE
  - Trusted App (TA)
  - Trusted App (TA)
  - TrustZone OS (TZOS)
- Secure Monitor

- Separate software stack
- Trusted applications (TAs)
- TrustZone OS (TZOS)
- TEE/REE
The “Hidden” Software Stack: TrustZone

- Separate software stack
- Trusted applications (TAs)
- TrustZone OS (TZOS)
- TEE/REE
- Basis for security: Has access to hardware keys
The “Hidden” Software Stack: TrustZone

- Separate software stack
  - Trusted applications (TAs)
  - TrustZone OS (TZOS)
  - TEE/REE
- Basis for security: Has access to hardware keys
- Access to TZ locked down: Only signed software can run
Problem: Dynamic analysis of TZ is hard!

Approach

Results: What did we learn?
Problem: Dynamic Analysis of TZ is Hard

Normal World/REE
- Android apps
- Android framework
- OS: Linux kernel

Secure World/TEE
- Trusted App (TA)
- Trusted App (TA)
- TrustZone OS (TZOS)
- Secure Monitor
- HW Crypto Keys
- Secure Peripherals
Problem: Dynamic Analysis of TZ is Hard

- Dynamic analysis needs ability to monitor target

![Diagram showing Normal World/REE vs Secure World/TEE]

- Android apps
- Android framework
- OS: Linux kernel
- Trusted App (TA)
- TrustZone OS (TZOS)
- Secure Monitor
- HW Crypto Keys
- Secure Peripherals
Problem: Dynamic Analysis of TZ is Hard

- Dynamic analysis needs ability to monitor target
- Debugging – needs memory/registers
- Feedback-driven fuzz testing – needs list of basic blocks covered
Problem: Dynamic Analysis of TZ is Hard

- Dynamic analysis needs ability to monitor target
  - Debugging – needs memory/registers
  - Feedback-driven fuzz testing – needs list of basic blocks covered
- However, cannot instrument TZ software or monitor TZ memory due to signing!
Problem: Dynamic Analysis of TZ is Hard

- Prior dynamic analysis approaches limited!
  - TA/TZOS binary reverse engineering
  - Fuzz testing without feedback
Solution: Dynamic Analysis By Emulation

• We build an emulator that runs real-world TZOSes and TAs
Solution: Dynamic Analysis By Emulation

• We build an emulator that runs real-world TZOSes and TAs
• Emulation enables dynamic analysis
  • Allows introspection and monitoring of TZ execution
Solution: Dynamic Analysis By Emulation

• We build an emulator that runs real-world TZOSes and TAs
• Emulation enables dynamic analysis
  • Allows introspection and monitoring of TZ execution
• We support four widely-used real-world TZOSes:
  • Qualcomm’s QSEE
  • Trustonic’s Kinibi
  • Samsung’s TEEGRIS
  • Linaro’s OP-TEE
Problem: Dynamic analysis of TZ is hard!

Approach: How did we run TZ in an emulator?

Results: What did we learn?
Challenge: Large Number of Components
Challenge: Large Number of Components

- Android Apps
- Android FW
- TEE Userspace
- Linux OS
- TEE Driver
- Hypervisor

Software
Challenge: Large Number of Components

- Android Apps
- Trusted Apps
- Android FW
- TEE Userspace
- Linux OS
- TEE Driver
- Hypervisor
- TrustZone OS
- Secure Monitor
- Boot ldr

Software
Challenge: Large Number of Components

Android Apps
Android FW
Linux OS
Hypervisor

TEE Userspace
TEE Driver

Secure Monitor

Trusted Apps
TrustZone OS
Boot ldr

Software
Hardware

Camera
Touch-screen
Fingerprint
System-on-Chip
Crypto HW TZASC/TZPC
Storage
Challenge: Large Number of Components

Android Apps
- Android FW
- Linux OS
- Hypervisor

TEE User-space
- TEE Driver

TrustZone OS
- Trusted Apps
- Boot ldr

Secure Monitor
- Camera
- Touch-screen
- Fingerprint
- System-on-Chip
- Crypto HW
- TZASC/TZPC
- Storage
Traditional Approach: Emulate all HW

- Android Apps
- Android FW
- Linux OS
- Hypervisor
- TEE Userspace
- TEE Driver
- TrustZone OS
- Trusted Apps
- Boot ldr

Software
Hardware

Reuse Unmodified Software Components
Traditional Approach: Emulate all HW

Android Apps
- Android FW
  - Linux OS
    - Hypervisor

TEE Userspace
- TEE Driver

TrustZone OS
- Trusted Apps
  - Bootldr

Secure Monitor

Hardware
- Camera
- Touchscreen
- Fingerprint
- System-on-Chip
- Crypto HW TZASC/TZPC
- Storage

Software
- Reuse Unmodified Software Components
- Emulate All Hardware Components
Traditional Approach: Emulate all HW

Impractical to emulate all hardware
Our Approach: Emulate Subset of HW and SW
Our Approach: Emulate Subset of HW and SW

- Android Apps
  - Android FW
  - Linux OS
  - TEE Userspace
  - TEE Driver
  - Hypervisor

- Trusted Apps
  - Boot ldr
  - TrustZone OS
  - Secure Monitor

- Hardware
  - Camera
  - GPS
  - Touch-screen
  - Fingerprint
  - System-on-Chip
  - Crypto HW
  - TZASC/TZPC
  - Storage

- Software

Study Component Dependencies
Our Approach: Emulate Subset of HW and SW

- Android Apps
  - Android FW
  - Linux OS
  - Hypervisor

- TEE Userspace
  - TEE Driver
  - Secure Monitor

- Trusted Apps

- Boot ldr

- Crypto HW TZASC/TZPC
  - Storage

- Camera
- Touchscreen
- Fingerprint
- System-on-Chip

Study Component Dependencies
Our Approach: Emulate Subset of HW and SW

Android Apps
  └── Android FW
    └── Linux OS
      └── Hypervisor

TEE Userspace
  └── TEE Driver
    └── TrustZone OS
      └── Boot ldr

Secure Monitor
  └── Camera
    └── Touch-screen
      └── Finger-print
        └── System-on-Chip
          └── Crypto HW
            └── TZASC/TZPC

Storage

Study Component Dependencies
Our Approach: Emulate Subset of HW and SW

- Android Apps
- Android FW
- Linux OS
- Hypervisor
- TEE Userspace
- TEE Driver
- TrustZone OS
- Trusted Apps
- Boot ldr
- Secure Monitor
- Camera
- Touch-screen
- Finger-print
- System-on-Chip
- Crypto HW TZASC/TZPC
- Storage

Study Component Dependencies

Tightly coupled, difficult to emulate dependency
Our Approach: Emulate Subset of HW and SW

- Android Apps
  - Android FW
  - Linux OS
  - Hypervisor

- TEE Userspace
  - TEE Driver
  - Secure Monitor

- Trusted Apps
  - TrustZone OS
  - Boot ldr

- Storage
  - Camera
  - Touch-screen
  - Finger-print
  - System-on-Chip
  - Crypto HW
    - TZASC/TZPC

Study Component Dependencies

Tightly coupled, difficult to emulate dependency
Our Approach: Emulate Subset of HW and SW

Loosely coupled, easy to emulate dependency

Tightly coupled, difficult to emulate dependency
Our Approach: Emulate Subset of HW and SW

- Android Apps
  - Android FW
    - Linux OS
      - TEE Userspace
        - TEE Driver
          - TrustZone OS
            - Bootldr
              - Emulate Bootloader Using Stub
              - Exclude Storage

- Trusted Apps
- Hypervisor
- Secure Monitor
  - Camera
  - Touch-screen
  - Fingerprint
  - System-on-Chip
  - Crypto HW
    - TZASC/TZPC
  - Storage
Our Approach: Emulate Subset of HW and SW

- Android Apps
  - Android FW
  - Linux OS
    - Hypervisor
    - TEE Userpace
      - TEE Driver
      - TrustZone OS
        - Bootldr
          - Secure Monitor
            - Camera
            - Touch-screen
            - Fingerprint
            - System-on-Chip
            - Crypto HW
              - TZASC/TZPC
            - Storage
          - Emulate Bootloader Using Stub
          - Exclude Storage

- Trusted Apps
Our Approach: Emulate Subset of HW and SW

- Android Apps
  - Android FW
  - Linux OS
  - Hypervisor

- TEE Userspace
  - TEE Driver

- Trusted Apps
  - TrustZone OS
  - Boot ldr

- Secure Monitor
  - Camera
  - Touch-screen
  - Fingerprint
  - System-on-Chip
  - Crypto HW
  - TZASC/TZPC
  - Storage

Emulate Bootloader Using Stub
Exclude Storage
Emulation Effort Feasible Using Patterns
Emulation Effort Feasible Using Patterns

• Patterns to Emulate Hardware (MMIO Loads and Stores)

  # Constant read (CONSTANT_READ_REG)
  v = read(CONSTANT_READ_REG);
  if (v != VALID_VALUE)
      fail();

  # Read-write (READ_WRITE_REG)
  write(READ_WRITE_REG, v1);
  v2 = read(READ_WRITE_REG);
  if (v2 != v1)
      fail();

  # Increment (INCR_REG)
  v = read(INCR_REG);
  if (read(INCR_REG) < v)
      fail();

  # Poll (POLL_REG)
  while (read(POLL_REG) != READY);

  # Random (RAND_REG)
  v1 = read(RAND_REG)
  v2 = read(RAND_REG)
  if (v1 == v2)
      fail();

  # Shadow (SHADOW_REG1, SHADOW_REG2)
  # Commit (COMMIT_REG)
  # Target (TARGET_REG1, TARGET_REG2)
  write(SHADOW_REG1, v1)
  write(SHADOW_REG2, v2)
  write(COMMIT_REG, COMMIT_VALUE)
  v3 = read(TARGET_REG1)
  v4 = read(TARGET_REG2)
  if ((v1 != v3) or (v2 != v4))
      fail();
Emulation Effort Feasible Using Patterns

- Patterns to Emulate Hardware (MMIO Loads and Stores)

```c

#define Constant_read (CONSTANT_READ_REG)
v = read(CONSTANT_READ_REG);
if (v != VALID_VALUE)
    fail();

#define Read-write (READ_WRITE_REG)
write(READ_WRITE_REG, v);
v2 = read(READ_WRITE_REG);
if (v2 != v1)
    fail();

#define Increment (INCR_REG)
v = read(INCR_REG);
if (read(INCR_REG) < v)
    fail();

#define Poll (POLL_REG)
while (read(POLL_REG) != READY);

#define Random (RAND_REG)
v1 = read(RAND_REG)
v2 = read(RAND_REG)
if (v1 == v2)
    fail();

#define Shadow (SHADOW_REG1, SHADOW_REG2)
#define Commit (COMMIT_REG)
#define Target (TARGET_REG1, TARGET_REG2)
write(SHADOW_REG1, v)
write(SHADOW_REG2, v2)
write(COMMIT_REG, COMMIT_VALUE)
v3 = read(TARGET_REG1)
v4 = read(TARGET_REG2)
if ((v1 != v3) or (v2 != v4))
    fail();
```
Emulation Effort Feasible Using Patterns

- Patterns to Emulate Hardware (MMIO Loads and Stores)

```c
# Constant read (CONSTANT_READ_REG)
v = read(CONSTANT_READ_REG);
if (v != VALID_VALUE)
    fail();

# Read-write (READ_WRITE_REG)
write(READ_WRITE_REG, v1);
v2 = read(READ_WRITE_REG);
if (v2 != v1)
    fail();

# Increment (INCR_REG)
v = read(INCR_REG);
if (read(INCR_REG) < v)
    fail();

# Poll (POLL_REG)
while (read(POLL_REG) != READY);

# Random (RAND_REG)
v1 = read(RAND_REG);
v2 = read(RAND_REG);
if (v1 == v2)
    fail();

# Shadow (SHADOW_REG1, SHADOW_REG2)
# Commit (COMMIT_REG)
# Target (TARGET_REG1, TARGET_REG2)
write(SHADOW_REG1, v1);
write(SHADOW_REG2, v2);
write(COMMIT_REG, COMMIT_VALUE);
v3 = read(TARGET_REG1);
v4 = read(TARGET_REG2);
if ((v1 != v3) or (v2 != v4))
    fail();
```
Emulation Effort Feasible Using Patterns

- Patterns to Emulate Software APIs

<table>
<thead>
<tr>
<th>Category</th>
<th>Difficulty</th>
<th>K</th>
<th>Q</th>
<th>T</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Emulated Boot Information Structure</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constants</td>
<td>Low</td>
<td>13</td>
<td>8</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Any value</td>
<td>Low</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Simple value</td>
<td>Low</td>
<td>2</td>
<td>1</td>
<td>14</td>
<td>2</td>
</tr>
<tr>
<td>Complex values</td>
<td>High</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td>18</td>
<td>13</td>
<td>16</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Category</th>
<th>Difficulty</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Emulated Secure Monitor Calls</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Return simple value</td>
<td>Low</td>
<td>0</td>
<td>-</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>Return constant</td>
<td>Low</td>
<td>1</td>
<td>-</td>
<td>5</td>
<td>-</td>
</tr>
<tr>
<td>Store/retrieve values</td>
<td>Low</td>
<td>1</td>
<td>-</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>Control transfer</td>
<td>High</td>
<td>3</td>
<td>-</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td>5</td>
<td>-</td>
<td>12</td>
<td>-</td>
</tr>
</tbody>
</table>
Emulation Effort Feasible Using Patterns

- Patterns to Emulate Software APIs
Emulation Effort Feasible Using Patterns

- Patterns to Emulate Software APIs

<table>
<thead>
<tr>
<th>Category</th>
<th>Difficulty</th>
<th>K</th>
<th>Q</th>
<th>T</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Emulated Boot Information Structure</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constants</td>
<td>Low</td>
<td>13</td>
<td>8</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Any value</td>
<td>Low</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Simple value</td>
<td>Low</td>
<td>2</td>
<td>1</td>
<td>14</td>
<td>2</td>
</tr>
<tr>
<td>Complex values</td>
<td>High</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td>18</td>
<td>13</td>
<td>16</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Emulated Secure Monitor Calls</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Return simple value</td>
<td>Low</td>
<td>0</td>
<td>-</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>Return constant</td>
<td>Low</td>
<td>1</td>
<td>-</td>
<td>5</td>
<td>-</td>
</tr>
<tr>
<td>Store/retrieve values</td>
<td>Low</td>
<td>1</td>
<td>-</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>Control transfer</td>
<td>High</td>
<td>3</td>
<td>-</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td>5</td>
<td>-</td>
<td>12</td>
<td>-</td>
</tr>
</tbody>
</table>
Implementation

QEMU
Implementation

QEMU

HW Emulation
Implementation
Implementation

QEMU

SW Emulation

TAs

TZOS

PartEmu Module

AFL Fuzz Module

Debug Module

QEMU

HW Emulation

Guest

TAs

TZOS
Implementation

- QEMU
- HW Emulation
- SW Emulation
- TAs
- TZOS
- PartEmu Run Management API
- QEMU Module
- AFL Fuzz Module
- Debug Module
- PartEmu Module API
- SW Emulation
- TAs
- TZOS
- Driver
- Guest

PartEmu Run Management API
Problem: Dynamic analysis of TZ is hard!

Approach: How did we run TZ in an emulator?

Results: What did we learn?
Fuzz Testing TAs Using AFL

16 Firmware Images
Fuzz Testing TAs Using AFL

- **16** Firmware Images
- **12** Smartphone / IoT vendors
Fuzz Testing TAs Using AFL

- **16** Firmware Images
- **12** Smartphone / IoT vendors
- **196** Unique TAs
Fuzz Testing TAs Using AFL

- **16** Firmware Images
- **12** Smartphone / IoT vendors
- **196** Unique TAs
- AFL Crashed **48** TAs
Fuzz Testing TAs Using AFL

- 16 Firmware Images
- 12 Smartphone / IoT vendors
- 196 Unique TAs
- AFL Crashed 48 TAs

- Found TZ-specific coding anti-patterns that led to crashes
Anti-Pattern 1:
Assumptions about Request Sequence
Anti-Pattern 1: Assumptions about Request Sequence

• TAs split work into small units → receive a sequence of requests
Anti-Pattern 1: Assumptions about Request Sequence

• TAs split work into small units → receive a sequence of requests

```c
char *ptr = NULL; // global
...
switch (request) {
  case INIT:
    init(ptr);
    break;
  case DO_ACTION:
    do_action(ptr);
    break;
  case UNINIT:
    uninit(ptr);
    break;
}
```
Anti-Pattern 1: Assumptions about Request Sequence

• TAs split work into small units $\rightarrow$ receive a sequence of requests

```c
char *ptr = NULL; // global
...
switch (request) {
    case INIT:
        init(ptr);
        break;
    case DO_ACTION:
        do_action(ptr);
        break;
    case UNINIT:
        uninit(ptr);
        break;
}
```
Anti-Pattern 1:
Assumptions about Request Sequence

• TAs split work into small units \(\rightarrow\) receive a sequence of requests

```c
char *ptr = NULL; // global
...
switch (request) {
    case INIT:
        init(ptr);
        break;
    case DO_ACTION:
        do_action(ptr);
        break;
    case UNINIT:
        uninit(ptr);
        break;
}
```
Anti-Pattern 1: Assumptions about Request Sequence

- TAs split work into small units → receive a sequence of requests

```c
char *ptr = NULL; // global
...
switch (request) {
  case INIT:
    init(ptr);
    break;
  case DO_ACTION:
    do_action(ptr);
    break;
  case UNINIT:
    uninit(ptr);
    break;
}
```
Anti-Pattern 1:
Assumptions about Request Sequence

• TAs split work into small units \( \rightarrow \) receive a sequence of requests

```c
char *ptr = NULL; // global
...
switch (request) {
    case INIT:
        init(ptr);
        break;
    case DO_ACTION:
        do_action(ptr);
        break;
    case UNINIT:
        uninit(ptr);
        break;
}
```
Anti-Pattern 1: Assumptions about Request Sequence

- TAs split work into small units → receive a sequence of requests

```c
char *ptr = NULL; // global
...
switch (request) {
    case INIT:
        init(ptr);
        break;
    case DO_ACTION:
        do_action(ptr);
        break;
    case UNINIT:
        uninit(ptr);
        break;
}
```

1. Null-pointer dereference
Anti-Pattern 1: Assumptions about Request Sequence

- TAs split work into small units → receive a sequence of requests

```c
char *ptr = NULL; // global
...
switch (request) {
    case INIT:
        init(ptr);
        break;
    case DO_ACTION:
        do_action(ptr);
        break;
    case UNINIT:
        uninit(ptr);
        break;
}
```

1. Null-pointer dereference

TA should properly handle any sequence of requests from CA
Anti-Pattern 2: Unvalidated Normal-World Pointers

Normal World/REE

Client App (CA)

Secure World/TEE

Trusted App (TA)
Anti-Pattern 2: Unvalidated Normal-World Pointers

- Normal World/REE
  - Client App (CA)
- Secure World/TEE
  - Trusted App (TA)
- Shared Memory
Anti-Pattern 2: Unvalidated Normal-World Pointers

Normal World/REE

Client App (CA)

Secure World/TEE

Trusted App (TA)

shm_ta_base

Shared Memory
Anti-Pattern 2: Unvalidated Normal-World Pointers

Diagram:
- Normal World/REE
  - Client App (CA)
  - Shared Memory
- Secure World/TEE
  - Trusted App (TA)
  - shm_ta_base
Anti-Pattern 2: Unvalidated Normal-World Pointers

Normal World/REE

Client App (CA)

ptr = off + shm_ta_base

Secure World/TEE

Trusted App (TA)

shm_ta_base

Shared Memory
Anti-Pattern 2: Unvalidated Normal-World Pointers

Client App (CA) \[ \text{ptr} = \text{off} + \text{shm\_ta\_base} \]

Trusted App (TA) \[ \text{shm\_ta\_base} \]

Shared Memory
Anti-Pattern 2: Unvalidated Normal-World Pointers

Client App (CA)

Secure World/TEE

Trusted App (TA)

Normal World/REE

shm_ta_base

ptr = off + shm_ta_base

mal_ptr

ptr
Anti-Pattern 2: Unvalidated Normal-World Pointers

Normal World/REE

Client App (CA)

ptr = off + shm_ta_base

Secure World/TEE

Shared Memory

shm_ta_base

ptr

Trusted App (TA)

mal_ptr

TA Memory leak / corruption
Anti-Pattern 2: Unvalidated Normal-World Pointers

TA should check that CA-supplied pointers point to shared memory
Anti-Pattern 3: Unvalidated Parameter Types

- GlobalPlatform TEE API allows 4 parameters in TA calls
  - Each parameter can be either a value or a pointer to a buffer

```c
TEE_Result TA_InvokeCommandEntryPoint(void *session, uint32_t cmd,
                                           uint32_t paramTypes, TEE_Params params[4])
{
    // Use params[0] as a buffer
    request_ptr = (struct request_struct *) params[0];
    switch (request_ptr->request) {
        ...
    }
}
```
Anti-Pattern 3: Unvalidated Parameter Types

• GlobalPlatform TEE API allows 4 parameters in TA calls
  • Each parameter can be either a value or a pointer to a buffer

```c
TEE_Result TA_InvokeCommandEntryPoint(void *session, uint32_t cmd, uint32_t paramTypes, TEE_Params params[4])
{
    // Use params[0] as a buffer
    request_ptr = (struct request_struct *) params[0];
    switch (request_ptr->request) {
        ...
    }
}
```
Anti-Pattern 3: Unvalidated Parameter Types

- GlobalPlatform TEE API allows 4 parameters in TA calls
  - Each parameter can be either a value or a pointer to a buffer

```c
TEE_Result TA_InvokeCommandEntryPoint(void *session, uint32_t cmd,
                                       uint32_t paramTypes, TEE_Params params[4])
{
    // Use params[0] as a buffer
    request_ptr = (struct request_struct *) params[0];
    switch (request_ptr->request) {
        ...
    }
}
```

```c
paramType(0) = TEEC_MEMREF;
```
Anti-Pattern 3: Unvalidated Parameter Types

- GlobalPlatform TEE API allows 4 parameters in TA calls
  - Each parameter can be either a value or a pointer to a buffer

```c
TEE_Result TA_InvokeCommandEntryPoint(void *session, uint32_t cmd,
                                       uint32_t paramTypes, TEE_Params params[4])
{
    // Use params[0] as a buffer
    request_ptr = (struct request_struct *) params[0];
    switch (request_ptr->request) {
        ...
    }

    paramTypes[0] = TEEC_MEMREF;
    paramTypes[0] = TEEC_VALUE;
```
Anti-Pattern 3: Unvalidated Parameter Types

• GlobalPlatform TEE API allows 4 parameters in TA calls
  • Each parameter can be either a value or a pointer to a buffer

```c
TEE_Result TA_InvokeCommandEntryPoint(void *session, uint32_t cmd, uint32_t paramTypes, TEE_Params params[4])
{
    // Use params[0] as a buffer
    request_ptr = (struct request_struct *) params[0];
    switch (request_ptr->request) {
    ...
    }
}
```

paramTypes[0] = TEEC_MEMREF;

TA Memory leak / corruption
Anti-Pattern 3: Unvalidated Parameter Types

- GlobalPlatform TEE API allows 4 parameters in TA calls
  - Each parameter can be either a value or a pointer to a buffer

```c
TEEResult TA_InvokeCommandEntryPoint(void *session, uint32_t cmd,
    uint32_t paramTypes, TEE_Parms params[4])
{
    // Use params[0] as a buffer
    request_ptr = (struct request_struct *) params[0];
    switch (request_ptr->request) {
        ...
    }
}
```

```
paramTypes(0) = TEEC_MEMREF;
paramTypes(0) = TEEC_VALUE;
```

*TA should check CA-supplied parameter types*
Conclusion

• We showed that it is **practically feasible** to run real-world TZOSes and TAs in an emulator
Conclusion

• We showed that it is *practically feasible* to run real-world TZOSes and TAs in an emulator

• Large-scale fuzz testing of TAs using the emulator found several *previously unknown* vulnerabilities in TAs with high reproducibility
Conclusion

• We showed that it is **practically feasible** to run real-world TZOSes and TAs in an emulator

• Large-scale fuzz testing of TAs using the emulator found several **previously unknown** vulnerabilities in TAs with high reproducibility

• We identified vulnerability patterns **unique to TA development**
  • Pointing to the need for **TZ-specific developer education**
Conclusion

• We showed that it is \textit{practically feasible} to run real-world TZOSes and TAs in an emulator

• Large-scale fuzz testing of TAs using the emulator found several \textit{previously unknown} vulnerabilities in TAs with high reproducibility

• We identified vulnerability patterns \textit{unique to TA development}
  • Pointing to the need for \textit{TZ-specific developer education}

Thank you!