vTZ: Virtualizing ARM TrustZone

Zhichao Hua, Jinyu Gu, Yubin Xia, Haibo Chen, Binyu Zang, Haibing Guan
Can VMs use TrustZone?
ARM TrustZone

- Two isolated execution environments
- Different worlds switch to each other by “smc” instruction
- Hardware resources can be partitioned into secure/non-secure part dynamically
  - Secure world can access all
  - Normal world can only access non-secure part
TrustZone Usage

• TrustZone on *mobile phone*
  – Secure storage, key protection, kernel integrity checker, malware detector, etc.

• TrustZone on *ARM server*
  – Has similar scenarios and requirements
TrustZone + Virtualization

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Ideal</th>
<th>Reality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Each guest VM has a secure world</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Guest VM can choose its own trust OS</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Isolation between different guests’ secure worlds</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Hardware resources partition between guest’s normal world and secure world</td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
## TrustZone + Virtualization

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Ideal</th>
<th>Reality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Each guest VM has a secure world</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Guest VM can choose its own trust OS</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Isolation between different guests’ secure worlds</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Hardware resources partition between guest’s normal world and secure world</td>
<td>✓</td>
<td>✗</td>
</tr>
</tbody>
</table>

The OS in secure world becomes single point of failure

Goals

• **Multiplexing** the secure world for guest VMs
  – Each guest VM can choose its own trust OS
  – Isolate each guest’s secure world
  – Provide hardware resources partition for each guest

• **Compatibility** for existing software in secure world
  – Provide same functionalities and interfaces with real TrustZone
  – Support existing trust OS

• **Minimizing TCB** of the new architecture
Alternative Designs

Design Choice I

- T-VMM (trusted VMM) in the secure world
  - Virtualize guest secure world with $\text{VM}_S$
- VMM in the normal world
  - Virtualize guest normal world with $\text{VM}_N$
- No virtualization extension in secure world
  - Both guest’s trust OS and TA run in user mode
  - ARM has virtualization unfriendly instructions
    - Trust OS needs to be modified
- System TCB
  - T-VMM (~10K of LoC)
  - VMM (millions of LoC)
Alternative Designs

**Design Choice I**
- Normal World
  - VM\(_N\)
  - APP
  - OS
  - VMM
- Secure World
  - VM\(_S\)
  - T-OS
  - TA
  - T-VMM

**Design Choice II**
- Normal World
  - VM\(_N\)
  - APP
  - OS
  - T-OS
  - VMM
- Secure World
  - VM\(_S\)
  - TA
  - T-VMM

- • Large TCB
- • No compatibility
- • Bad performance
- • Large TCB
- • Has compatibility
- • Good performance

- • Virtualize guest secure world in real normal world
- • Secure world is not used
- • System TCB
  - – VMM (millions of LoC)
Alternative Designs

Design Choice I
- Normal World
  - VM_N
    - APP
    - OS
    - VMM
- Secure World
  - VM_S
    - T-OS
    - TA
    - T-VMM

• Large TCB
• No compatibility
• Bad performance

Design Choice II
- Normal World
  - VM_N
    - APP
    - OS
    - VMM
- Secure World
  - VM_S
    - T-OS
    - TA

• Large TCB
• Has compatibility
• Good performance

Design Choice III
- Normal World
  - VM_N
  - OS
  - VMM
- Secure World
  - VM_S
  - T-OS

• Small TCB
• Has compatibility

TCB: TCB
APP: APP
TA: TA
VMM: VMM
VM_N: VM_N
VM_S: VM_S
OS: OS
T-OS: T-OS
Secured Modules: CIEEs
CFLock: CIEEs
Threat Model

- Any guest may be an attacker
- VMM is buggy, can be compromised
- Code integrity is protected during system boot by secure boot technology
- Side-channel attacks and physical attacks are not considered
Overview

• Emulate guest normal world/secure world with different VMs
  – World switching is performed by switching between two VMs

• SMM (Secured Memory Mapping)
  – Memory mapping

• SWS (Secured World Switching)
  – CPU Context

• CIEE (Constrained Isolated Execution Environment)
  – Protect critical logic in hyp mode

• CFLock (Control Flow Lock)
### Properties need to be enforced by vTZ

<table>
<thead>
<tr>
<th>TrustZone Features</th>
<th>System Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secure Boot</td>
<td>SW must boot before NW.</td>
</tr>
<tr>
<td></td>
<td>Boot image of SW must be checked.</td>
</tr>
<tr>
<td></td>
<td>SW cannot be replaced.</td>
</tr>
<tr>
<td>CPU States Protection</td>
<td>“smc” must switch to the correct world.</td>
</tr>
<tr>
<td></td>
<td>Protect the integrity of NW CPU states during switching.</td>
</tr>
<tr>
<td></td>
<td>Protect SW CPU states.</td>
</tr>
<tr>
<td>Memory Isolation</td>
<td>Only SW can access secure memory.</td>
</tr>
<tr>
<td></td>
<td>Only SW can configure memory partition.</td>
</tr>
<tr>
<td>Peripheral Assignment</td>
<td>Secure interrupts must be injected into SW.</td>
</tr>
<tr>
<td></td>
<td>NW cannot access secure peripherals.</td>
</tr>
<tr>
<td></td>
<td>Secure peripherals are trusted for SW</td>
</tr>
<tr>
<td></td>
<td>Only SW can partition interrupt/peripherals.</td>
</tr>
</tbody>
</table>
## Properties need to be enforced by vTZ.

<table>
<thead>
<tr>
<th>TrustZone Features</th>
<th>System Properties</th>
</tr>
</thead>
</table>
| Secure Boot        | SW must boot before NW.  
                      | Boot image of SW must be checked.  
                      | SW cannot be replaced. |
| CPU States Protection | **“smc” must switch to the correct world.**  
                           | Protect the integrity of NW CPU states during switching.  
                           | Protect SW CPU states. |
| Memory Isolation   | **Only SW can access secure memory.**  
                           | **Only SW can configure memory partition.** |
| Peripheral Assignment | Secure interrupts must be injected into SW.  
                           | NW cannot access secure peripherals.  
                           | Secure peripherals are trusted for SW  
                           | Only SW can partition interrupt/peripherals. |
P1. Only secure world can access secure memory

• Challenge
  – Untrusted VMM controls all memory mappings
  – Map one guest’s secure memory to its normal world or to another guest
  – Map one guest’s secure memory to VMM’s address space

• Solution
  – SMM exclusively controls all memory mappings to physical memory
  – SMM checks memory mappings
Two kinds of mappings to the physical memory
- Stage-2 page table maps guest physical address to physical address
- Hyp page table maps virtual address to physical address for VMM

Three ways memory mapping can be modified
- Enabling a page table
- Disabling the address translation
- Changing the entries of page table

Only secure world can access secure memory

```assembly
.enable_guest_pt:
  ldr x1, [x0]  // load page table
  MSR VTTBR_EL2, x1  // enable table

.enable_hyp_pt:
  ldr x1, [x0]  // load page table
  MSR TTBR0_EL2, x1  // enable table

.set_pte:
  str x1, [x0]  // change table entry
```
SMM (Secured Memory Mapping)

- SMM exclusively controls the mapping
  - Replace all page table maintain instructions
  - Enforce hypervisor’s code as read-only (R.O.)
  - Enforce page table as read-only
- SMM enforces memory mapping policy
  - E.g., one guest’s secure memory can only be mapped to its own secure world

P1. Only secure world can access the secure memory
P2. “smc” must switch to the correct world

• Challenge
  – Untrusted VMM controls the scheduleing of all VMs
  – “smc” may switch to a malicious VM
  – “smc” may switch to a wrong entry of guest secure world
  – Untrusted VMM may tamper with the CPU context during switching

• Solution
  – SWS hooks all switching between a VM and the VMM
  – SWS checks all switching
SWS (Secured World Switching)

- SWS Interposes switching between a VM and the VMM
  - **VM_exit** is triggered by exception, hooked by CFLock
  - **VM_enter** is performed by special instructions, replacing all of them
    - E.g., *eret*

- SWS binds each guest’s secure world and normal world
  - Identify VMs presenting guest’s two worlds by VMID
CFLock (Control Flow Lock)

- CFLock: hooks the control flow of exception handling
- Ensure the integrity of vector table containing exception handlers
  - Replace instructions which modify vector table base register
  - Mark vector table as read-only
- Add hook in each exception handler

```
.text
Init_vector:
  ldr x1, [x0] // load vector table
  MSR VBAR_EL2, x1 // enable vector table
  invoke_SMM // enable vector table

handler1:
  b CFLock...
```

Vector Table:
- `b handler1`
- `b handler2`
- `b handler3`
- `b handler4`
- `b handler5`
P2. “smc” must switch to the correct world

- Untrusted VMM registers guest’s two VMs in SWS
  - SWS only allows entering a registered VM
- Untrusted VMM schedules all VMs
- SWS checks all VM_enter / VM_exit operations
P3. Only secure world can partition memory

- Guest configures memory partition in its secure world by accessing a memory partition device (TZASC)

- Challenge
  - There is only one TZASC
  - Cannot control TZASC by the untrusted VMM

- Solution
  - Providing trusted virtual TZASC by “trap and emulate”
P3. Only secure world can partition memory

- **CFLock** traps memory accesses of virtual TZASC
  - Memory mapped device
  - Accessing triggers page fault exception

- **SWS** identifies current VM
  - Only guest secure world can do the partitioning

- Need an isolated execution environment to emulate device
  - Reuse some structure of the VMM
  - Protected from VMM
  - Not included in system TCB
P3. Only secure world can partition memory

• **CIEE** (Constrained Isolated Execution Environment)

• Protects a piece of code in the hyp mode

• Excluded from system TCB
CIEE (Constrained Isolated Execution Environment)

• Located in the hyp mode

• Prevent against the untrusted VMM
  – Single entry point
  – Run-to-completion
  – No dependence on the hypervisor’s data
  – No data exposure to the hypervisor
  – Unforgeable to the secure world

• Exclude CIEE from system TCB
  – Isolated privilege
CIEE (Constrained Isolated Execution Environment)

- CIEE components
  - CIEE-Code
  - Stack page
  - Secure object

- Enforce five security policies

- Isolated privilege
  - Diff privilege for each CIEE
  - Diff copy of secure obj. for each guest
P3. Only secure world can partition memory

- **CFLock** traps memory accesses of virtual TZASC
  - Memory mapped device
  - Accessing triggers page fault exception

- **SWS** identifies current VM
  - Only guest secure world can do the partitioning

- Emulate TZASC in an **CIEE**
Evaluation

• Can vTZ support existing trust OS?

• How is the performance of server applications on vTZ?

• How is the performance of application with multiple VMs?
Evaluation

• Hardware platform
  – Hikey (ARMv8) with eight 1.2 GHz cores
  – Exynos (ARMv7) with one 1.7GHz core

• Software environment
  – Xen 4.4 + Linux 4.1
Compatibility

• Port two existing trust OSes on vTZ
  – seL4 [SOSP’2009]
  – OP-TEE [https://github.com/OP-TEE/]

• Porting effort
  – Add description file to describe the device base addresses
    • Base addresses of memory and devices
    • Same as porting an OS on an ARM SoC
Application Overhead

Throughput of Apache (MB/s) vs. Size of TCP buffer (Bytes):
- Native
- TZ
- Xen
- vTZ

Throughput of MongoDB (Kops/s) vs. Size of item value (Bytes):
- Native
- TZ
- Xen
- vTZ
Application Overhead

GoHttps on ARMv7(a) and ARMv8(b) with different VMs.

- Native (N)
- TrustZone (TZ)
- Xen (X)
- Our system (vTZ)
Conclusion

- Analyze security properties of TrustZone
- Combine TrustZone and virtualization to multiplex secure world for each guest
  - **SMM** exclusively controls the memory mapping
  - **CFLock** hooks all exceptions in the hyp mode
  - **SWS** checks all switching between a VM and the VMM
  - **CIEEs** to protect pieces of code in the hyp mode and exclude them from system TCB
- Small system TCB
- Compatible with existing trust OS
  - Porting two trust OSes on vTZ
- Acceptable performance overhead
Thanks

Institute of Parallel And Distributed Systems (IPADS)
http://ipads.se.sjtu.edu.cn