Topic 10. Trusted Execution Environment

Instructor: Dr. Kun Sun
Outline

• Introduction of TEE

• Cache Incoherency in ARM TrustZone
  1. For bad: CacheKit – cache based rootkit
  2. For good: CaSE – prevent cold boot attack & malicious OS

• Future Research Directions
Outline

• Introduction of TEE
• Cache Incoherency in ARM TrustZone
  1. For bad: CacheKit – cache based rootkit
  2. For good: CaSE – prevent cold boot attack & malicious OS
• Future Research Directions
Trusted Execution Environment (TEE)

- Mobile payment and banking
  - Account information and passcode protection

- BYOD (Bring your own device)
  - Personal vs. Business

- Protection against potential attacks
  - Malware compromises OS
TEE vs. Traditional Security

TEE

• Isolated Computing Environment
• Secure Storage
• Remote Attestation
• Secure Provisioning
• Trusted Path

Traditional security

• Confidentiality
• Integrity
• Availability
ARM TrustZone

• Two isolated domains
  – Secure/non-secure CPU States
  – Two virtual MMU
  – Memory isolation
  – Device isolation
  – TrustZone-Aware ARM generic interrupt controller (GIC)

• Commercial Application
  – GlobalPlatform API standards
  – Trustonic Trusted Execution Environment (TEE)
  – Samsung Knox
Cache Incoherence in TrustZone

- The cache contents of the two worlds, secure and non-secure, potentially being different even when they are mapped to the same physical address.

- The secure world cannot access the CPU cache for the normal world. Similarly, the normal world cannot access the CPU cache for the secure world either.
Based on this observation

- CacheKit: Evading Memory Introspection Using Cache Incoherence
  - A new type of rootkit

- CaSE: Cache-Assisted Secure Execution on ARM Processors
  - An isolated computing environment in adversarial environment
    - Cold boot attacks
    - Malicious OS
CacheKit: Evading Memory Introspection Using Cache Incoherence [EuroS&P 16]
There are two main goals of rootkit:

give attacker access to the system
concealing their presence in the system
Rootkit in Attack Cycle
## Good Old Days - Rootkit

<table>
<thead>
<tr>
<th>Category</th>
<th>time</th>
<th>example</th>
<th>Detection Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>App Level File Based</td>
<td>90s</td>
<td>Login, Ls</td>
<td>OS – File Checksum</td>
</tr>
<tr>
<td>OS level Mem based</td>
<td>mid 90s</td>
<td>Adore, Fu</td>
<td>Hypervisor – VMI</td>
</tr>
<tr>
<td>Below OS Level Hw Based</td>
<td>2000s</td>
<td>Blue Pill, SMMR</td>
<td>TEE – Physical Mem Check</td>
</tr>
</tbody>
</table>
System Introspection

- Using a high privilege entity in the system or outside the system to examine the state of the computing system.

Diagram:
- Introspection
- Application
- OS
- VMM
- TrustZone / TEE
- Higher Privilege
Two Directions for Rootkit

1. **get to lower position in the sw stack to avoid detection**
   - **VMMRoot** by Joanna Rutkowska (Blackhat 06)
   - Also known as blue pill attack, a malicious hypervisor is inserted into below the victim OS to manipulate states.
   - **SMMR** (SecureComm, phrack, 2008)
   - BIOS firmware is modified to add malicious logic to system management mode software in x86 systems.
   - **Adore-Phrack**
   - DKOM for file hiding, process hiding

Bigger Guns
- Higher Privilege

VMM Rootkit – Blue Pill
SMM Rootkit
Adore
Two Directions for Rootkit

• **Manipulate hardware features to hide malicious activity**

  - **Cloacker** (Oakland 2008)
    - Changes ARM registers to redirect the location of exception vector

  - **Shadow Walker** (Blackhat 05)
    - Exploits split TLB in x86

  - **HiVE** (ASIACCS 15)
    - Use IORR register to mask memory area with asynchronous read / write destination

**Better Hiding Tactic**
- Leaf Gecko

Cloacker
ShadowWalker
HiVE
Intuition

• We can’t have greater power, so we better hide ...

• Accurate memory acquisition is the foundation of system introspection

• Wouldn’t it be nice to hide from the memory acquisition?
How is Memory Captured

CPU Memory Access Order

DMA

register

L1 Cache

L2 Cache
ARM TrustZone Technology

- It is a **full system** security extension, that extends to a **protected domain of memory, I/O devices and processors**.
Security Configuration Register (SCR) has NS bit indicates which world
Background - Caching in TrustZone
Same PA different Values

Secure World
- Introspector
  VA: 0x2465
- Secure System Page Table
- Non-Secure PA: 0x1234

Non-Secure World
- Rich OS
  VA: 0xA563
- Non-Secure System Page Table
- Non-Secure PA: 0x1234

Processor Cache
- PA: 0x1234, NS = 0, Line = 0x3491
- Locked
- PA: 0x1234, NS = 1, Line = 0xBF00

North Bridge Mux
- I/O Address Space
- RAM Address Space
- DMA-based Memory Forensics
Problem

- Incoherent cache -> Yes
- Cache is also dynamic, i.e., it will be evicted in no time
ARM offers cache locking to improve performance of certain specialized system design.

The cache locking is enabled by modifying the eviction policy of certain entries to never evict.

Granularity of cache lock in Cortex-A8 is cache way for the 8 way 512 set L2 cache

We use cache locking to lock down the incoherent cache.
Our Approach to Hide from TZ
Exploiting the cache incoherence

• Absence from RAM – Incoherent Cache Only
  – Use Cache-as-RAM to load malicious code only in cache
  – Incoherent cache is not accessible by the Secure World

• Persist in Cache - Cache Locking
  – Hardware Cache Lock (L2 Cache Lockdown Register)
  – We also optimize our cache usage in L2 cache
Wait a sec – How about normal world?

• Using cache incoherence, we hide from TrustZone

• How do we run from the normal world memory acquisition?
ARM Physical Address Space

Physical Address Space

System Memory

I/O Space

0x80000000

0x15000000

0x13000000

0xFFFFF000
Cache Space in I/O Address Achievements

• Evading from Normal World
  – By hiding in a physical address that is not supposed to be read / write / ex

• Mitigate the risk of cache flush by writing contents out to the I/O address space which will simply drop all the data
Exploiting – Putting the two techniques together

Hide From Secure  Hide From Normal

Fig. 2. CacheKit Overview
CacheKit Prototype
Platform Details

- I.MX53 Freescale Development Board – with Cortex-A8 Processor
- 1GB DRAM, 32KB L1 Instruction Cache, 32KB L1 Data Cache, 256 KB L2 Cache
- L2 is 8 way, 512 set unified cache
- 2.6.33 Linux kernel
CacheKit Evaluation

• Effectiveness of Cachekit
  • Can we use this approach to hide our malicious code?

• Performance Impact
  • How much are we slowing down the system by monopolizing cache usage?
Effectiveness of CacheKit

• **Cache Existence** – exist in and only in cache
• **Cache Persistence** – cache contents actually stay in real system
• **Cache Elusiveness** – show that none of the current toolkit can acquire memory hidden by CacheKit
  • Verify that memory dump neither normal world nor secure world
  • Verify that DMA cannot dump the cache contents
Rootkit Paradox

• CacheKit leaves no trace in RAM.

• It can still be detected when the defender knows exactly where to find it using the processor. This fundamental conflict between concealment and presence is known as *Rootkit Paradox*.
  – Comparing the address translation performed by MMU and the address stored in the paging table.
  – Using debugging interface such as JTAG to look inside the processor cache.
Countermeasure

• Detect
  – L2 Cache Locks, incoherent TLB

• Clean up process
  – Remove hook
    • removes the exception vector hook
  – Flush Cache
    • removing the malicious codes in cache
Limitations

• Persistence Across Power States
  – Use of cache for program memory makes it difficult to persist across power states, tradeoff between persistence and stealth

• Insertion under Harden Environment
  – Under TZ kernel protection, it is hard to insert rootkits
CaSE: Cache-Assisted Secure Execution on ARM Processors [S&P 2016]
Smart Devices Going Mobile
Hardware Attacks - Cold Boot Attack
Multi-vector Adversary
Introducing CaSE - Goals

- Defense against Multi-Vector adversary
  - Physical memory disclosure attack – Cold boot
  - Compromised rich OS

- Provide confidentiality and integrity to the applications in TEE
Threat Model

System On Chip (SoC)

Processor Cache

Secure Cache

NonSecure Cache

Secure Memory

NonSecure Normal World Memory

Secure OS

NonSecure Rich OS

DRAM

Cold Boot Attack

Software Attack
Case-based Execution in Secure World

System On Chip (SoC)

DRAM

NonSecure Normal World Memory

NonSecure
Rich OS

Secure Memory

Secure OS

Cold Boot Attack

Software Attack
Evaluation of using Cache as Memory

<table>
<thead>
<tr>
<th>Application</th>
<th>Code+Data (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>2.4</td>
</tr>
<tr>
<td>RSA</td>
<td>10</td>
</tr>
<tr>
<td>SHA1</td>
<td>5</td>
</tr>
<tr>
<td>CaSE Crypto Lib</td>
<td>17.4</td>
</tr>
<tr>
<td>Kernel Integrity Checker</td>
<td>6.6</td>
</tr>
<tr>
<td>CaSE Packer</td>
<td>2.8</td>
</tr>
<tr>
<td>Packed CaSE Crypto Lib</td>
<td>20.4</td>
</tr>
<tr>
<td>Packed Kernel Checker</td>
<td>9.5</td>
</tr>
</tbody>
</table>
Summary of CaSE

• TrustZone enabled SoC-bound execution environment
  – protect against both cold boot attack and compromised rich OS attack.

• Demonstrate the practical usage of our system on real ARM platform.
  – build a crypto library and a kernel integrity checker
Future Research Directions

- Current development platform is single core system
  - We are studying the multi-core systems

- SGX vs. TrustZone

<table>
<thead>
<tr>
<th>Feature</th>
<th>SGX</th>
<th>TrustZone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>x86</td>
<td>ARM</td>
</tr>
<tr>
<td>RAM Encryption</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Secure Interrupt</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Side channel</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Host OS suspension</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>DoS attack</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>