Ninja: Towards Transparent Tracing and Debugging on ARM

Zhenyu Ning & Fengwei Zhang
Wayne State University
{zhenyu.ning, fengwei}@wayne.edu
Outline

• Introduction
• Background
• System Overview
• Evaluation
• Conclusion
Outline

• Introduction
• Background
• System Overview
• Evaluation
• Conclusion
Evasion Malware
Evasion Malware
Malware Analysis

Applications

Operating System

Hypervisor/Emulator
Malware Analysis

Applications

Operating System

Hypervisor/Emulator
Malware Analysis

Applications

Operating System

Hypervisor/Emulator

Limitation:

• Unarmed to anti-virtualization or anti-emulation techniques
Malware Analysis

Applications

Operating System

Hypervisor/Emulator

Malware Analyzer
Malware Analysis

Limitation:

- Unable to handle malware with high privilege (e.g., rootkits)
Malware Analysis

Applications

Operating System

Hypervisor/Emulator

Hardware

MalT
S&P 15
Malware Analysis

Limitations:

• High performance overhead on mode switch
• Unprotected modified registers
• Vulnerable to external timing attack
Transparency Requirements

• An *Environment* that provides the access to the states of the target malware

• An *Analyzer* which is responsible for the further analysis of the states
Transparency Requirements

- An *Environment* that provides the access to the states of the target malware
  - It is isolated from the target malware
  - It exists on an off-the-shelf (OTS) bare-metal platform
- An *Analyzer* which is responsible for the further analysis of the states
Transparency Requirements

• An **Environment** that provides the access to the states of the target malware
  • It is isolated from the target malware
  • It exists on an off-the-shelf (OTS) bare-metal platform

• An **Analyzer** which is responsible for the further analysis of the states
  • It should not leave any detectable footprints to the outside of the environment
Outline

• Introduction
• Background
• System Overview
• Evaluation
• Conclusion
Background - TrustZone

ARM TrustZone technology divides the execution environment into **secure** domain and **non-secure** domain.

- The RAM is partitioned to **secure** and **non-secure** region.
- The interrupts are assigned into **secure** or **non-secure** group.
- Secure-sensitive registers can only be accessed in secure domain.
- Hardware peripherals can be configured as secure access only.
Background - TrustZone

- In ARMv8 architecture, exceptions are delivered to different Exception Levels (ELs).
- The only way to enter the secure domain is to trigger a EL3 exception.
- The exception return instruction (ERET) can be used to switch back to the non-secure domain.
Background – PMU and ETM

• The Performance Monitor Unit (PMU) leverages a set of performance counter registers to count the occurrence of different CPU events.

• The Embedded Trace Macrocell (ETM) traces the instructions and data of the system, and output the trace stream into pre-allocated buffers on the chip.

• Both PMU and ETM exist on ARM Cortex-A5x and Cortex-A7x series CPUs, and do NOT affect the performance of the CPU.
Outline

- Introduction
- Background
- System Overview
- Evaluation
- Conclusion
Overview

Non-secure Domain

Rich OS

App

App

Target Malware
Overview

Non-secure Domain
- Rich OS
  - App
  - App
  - Target Malware

Secure Domain
- Secure Interrupt Handler

Secure Interrupt
Overview

Non-secure Domain

Rich OS

App

App

Target Malware

Secure Domain

Secure Interrupt

Secure Interrupt Handler

Trace Subsystem:

- Instruction Trace
- System Call Trace
- Android API Trace
Overview

Non-secure Domain

Rich OS

App

App

Target Malware

Secure Domain

Secure Interrupt

Secure Interrupt Handler

Trace Subsystem

Debug Subsystem

Debug Subsystem:

• Single Stepping

• Breakpoints

• Memory R/W
Overview

Non-secure Domain
- Rich OS
  - App
  - App
  - Target Malware

Secure Domain
- Secure Interrupt Handler
  - Secure Port
  - Trace Subsystem
  - Debug Subsystem

Remote Debugging Client

Wayne State University
Overview

Non-secure Domain

Rich OS

App

App

Target Malware

Secure Domain

Secure Interrupt Handler

Trace Subsystem

Debug Subsystem

Remote Debugging Client

Wayne State University

COMPASS LAB (HTTP://COMPASS.CS.WAYNE.EDU)
Hardware Traps

Non-secure Domain

<table>
<thead>
<tr>
<th>......</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRS X0, PMCR_EL0</td>
</tr>
<tr>
<td>MOV X1, #1</td>
</tr>
<tr>
<td>AND X0, X0, X1</td>
</tr>
<tr>
<td>......</td>
</tr>
</tbody>
</table>
Hardware Traps

Non-secure Domain

......
MRS X0, PMCR_EL0
MOV X1, #1
AND X0, X0, X1
......

Secure Domain

Analyzing the instruction

MDCR_EL3.TPM = 1
## Hardware Traps

<table>
<thead>
<tr>
<th>Non-secure Domain</th>
<th>Secure Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>......</td>
<td>Analyzing the instruction</td>
</tr>
<tr>
<td>MRS X0, PMCR_EL0</td>
<td>MOV X0, #0x41013000</td>
</tr>
<tr>
<td>MOV X1, #1</td>
<td></td>
</tr>
<tr>
<td>AND X0, X0, X1</td>
<td></td>
</tr>
<tr>
<td>......</td>
<td></td>
</tr>
</tbody>
</table>

The instruction `MDCR_EL3.TPM = 1` triggers a trap in the non-secure domain. The trap handler then analyses the instruction and takes appropriate action.
Hardware Traps

Non-secure Domain

......
MRS X0, PMCR_EL0
MOV X1, #1
AND X0, X0, X1
......

Secure Domain

MDCR_EL3.TPM = 1

Analyzing the instruction
MOV X0, #0x41013000
Modifying saved ELR_EL3
## Hardware Traps

### Non-secure Domain

<table>
<thead>
<tr>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>......</td>
</tr>
<tr>
<td>MRS X0, PMCR_EL0</td>
</tr>
<tr>
<td>MOV X1, #1</td>
</tr>
<tr>
<td>AND X0, X0, X1</td>
</tr>
<tr>
<td>......</td>
</tr>
</tbody>
</table>

### Secure Domain

- MDCR_EL3.TPM = 1
- Analyzing the instruction
  - MOV X0, #0x41013000
  - Modifying saved ELR_EL3
  - ERET
Outline

• Introduction
• Background
• System Overview
• Evaluation
• Conclusion
Evaluation - Transparency

• Environment:

• Analyzer:
Evaluation - Transparency

• Environment:
  ✓ Isolated

• Analyzer:
Evaluation - Transparency

• Environment:
  ✓ Isolated
  ✓ Exists on OTS platforms

• Analyzer:
Evaluation - Transparency

• Environment:
  ✓ Isolated
  ✓ Exists on OTS platforms

• Analyzer:
  ✓ No detectable footprints?
Evaluation - Transparency

- Environment:
  - ✔ Isolated
  - ✔ Exists on OTS platforms

- Analyzer:
  - ✔ No detectable footprints?

We believe that the hardware-based approach provides better transparency.

To build a fully transparent system, we may need additional hardware support.
Evaluation – Performance of the TS

• Testbed Specification
  
  • ARM Juno v1 development board
  
  • A dual-core 800 MHZ Cortex-A57 cluster and a quad-core 700 MHZ Cortex-A53 cluster
  
  • ARM Trusted Firmware (ATF) v1.1 and Android 5.1.1
Evaluation – Performance of the TS

- Calculating one million digits of $\pi$
- GNU Multiple Precision Arithmetic Library

<table>
<thead>
<tr>
<th>Tracing Type</th>
<th>Mean</th>
<th>STD</th>
<th>#Slowdown</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base: Tracing Disabled</td>
<td>2.133 s</td>
<td>0.69 ms</td>
<td></td>
</tr>
<tr>
<td>Instruction Tracing</td>
<td>2.135 s</td>
<td>2.79 ms</td>
<td>1x</td>
</tr>
<tr>
<td>System call Tracing</td>
<td>2.134 s</td>
<td>5.13 ms</td>
<td>1x</td>
</tr>
<tr>
<td>Android API Tracing</td>
<td>149.372 s</td>
<td>1287.88 ms</td>
<td>70x</td>
</tr>
</tbody>
</table>
## Evaluation – Performance of the TS

- Performance scores evaluated by CF-Bench

<table>
<thead>
<tr>
<th></th>
<th>Native Scores</th>
<th>Java Scores</th>
<th>Overall Scores</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>#Slowdown</td>
<td>Mean</td>
</tr>
<tr>
<td>Basic: Tracing Disabled</td>
<td>25380</td>
<td></td>
<td>18758</td>
</tr>
<tr>
<td>Instruction Tracing</td>
<td>25364</td>
<td>1x</td>
<td>18673</td>
</tr>
<tr>
<td>System call Tracing</td>
<td>25360</td>
<td>1x</td>
<td>18664</td>
</tr>
<tr>
<td>Android API Tracing</td>
<td>6452</td>
<td>4x</td>
<td>122</td>
</tr>
</tbody>
</table>
Evaluation – Domain Switching Time

• Time consumption of domain switching (in µs)

• 34x-1674x faster than MalT (11.72 µs)

<table>
<thead>
<tr>
<th></th>
<th>ATF Enabled</th>
<th>Ninja Enabled</th>
<th>Mean</th>
<th>STD</th>
<th>95% CI</th>
</tr>
</thead>
<tbody>
<tr>
<td>✗ ✗</td>
<td>✗ ✗</td>
<td>0.007</td>
<td>0.000</td>
<td>[0.007, 0.007]</td>
<td></td>
</tr>
<tr>
<td>✔ ✗</td>
<td>✔</td>
<td>0.202</td>
<td>0.013</td>
<td>[0.197, 0.207]</td>
<td></td>
</tr>
<tr>
<td>✔ ✔</td>
<td>✔</td>
<td>0.342</td>
<td>0.021</td>
<td>[0.334, 0.349]</td>
<td></td>
</tr>
</tbody>
</table>
Outline

• Introduction
• Background
• System Overview
• Evaluation
• Conclusion
Conclusion

• **Ninja**: A malware analysis framework on ARM.

• A debug subsystem and a trace subsystem

• Using TrustZone, PMU, and ETM to improve transparency

• The hardware-assisted trace subsystem is immune to timing attack.
Thank you!
Email: zhenyu.ning@wayne.edu

Questions?