Transparent Malware Debugging on x86 and ARM

Zhenyu Ning and Fengwei Zhang

COMPASS Lab
Wayne State University

April 27, 2018
Outline

- Introduction
- Background
- Towards Transparent Malware Analysis
  - MalT on x86 Architecture
  - Ninja on ARM Architecture
- Conclusions
Outline

- Introduction
- Background
- Towards Transparent Malware Analysis
  - MalT on x86 Architecture
  - Ninja on ARM Architecture
- Conclusions
What is transparent malware analysis?
Transparency

What is transparent malware analysis?

- Analyzing the malware without being aware.
- “Transparent” means that the malware cannot detect it.
Why transparency is important?
Evasive Malware

Computer System

Application

Malware
Evasive Malware
Evasive Malware

Analyzer

Computer System

Application

Application
What is the current state of malware analysis systems?
Malware Analysis
Malware Analysis

Application

Operating System

Hypervisor/Emulator

Malware Analyzer
Malware Analysis

- Unarmed to anti-virtualization or anti-emulation techniques.
- Large performance overhead.
Malware Analysis

- Application
  - App
  - App
  - Malware
- Operating System
  - Malware Analyzer
- Hypervisor/Emulator
Malware Analysis

Unable to handle malware with high privilege (e.g. rootkits).
Transparency Requirements

What makes a transparent malware analysis system?
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**
- Towards Transparent Malware Analysis
  - MalT on x86 Architecture
  - Ninja on ARM Architecture
- Conclusions
System Management Mode (SMM) [1] is special CPU mode existing in x86 architecture, and it can be used as a hardware isolated execution environment.

- Originally designed for implementing system functions (e.g., power management)
- Isolated System Management RAM (SMRAM) that is inaccessible from OS
- Only way to enter SMM is to trigger a System Management Interrupt (SMI)
- Executing RSM instruction to resume OS (Protected Mode)
TrustZone Technology

ARM TrustZone technology [2] divides the execution environment into a secure domain and a non-secure domain.

- The RAM is partitioned to secure and non-secure regions.
- The interrupts are assigned into the secure or non-secure group.
- Secure-sensitive registers can only be accessed in secure domain.
- Hardware peripherals can be configured as secure access only.
PMU and ETM

- The Performance Monitor Unit (PMU) [3, 4] leverages a set of performance counter registers to count the occurrence of different CPU events.

- The Embedded Trace Macrocell (ETM) [5] traces the instructions and data of the system, and output the trace stream into pre-allocated buffers on the chip.

- The PMU exists in both x86 and ARM architecture while the ETM is ARM special hardware.
Outline

- Introduction
- Background
- **Towards Transparent Malware Analysis**
  - MalT on x86 Architecture [S&P’15]
  - Ninja on ARM Architecture [USENIX Security’17]
- Conclusions
Towards Transparent Malware Analysis

- **Application**: App, App, Malware
- **Operating System**
- **Hypervisor/Emulator**: MalT on x86, Ninja on ARM
- **Hardware**
MalT on x86 Architecture

Debugging Client

1) Trigger SMI

2) Debug command

GDB-like Debugger

3) Response message

Debugging Server

SMI handler

Inspect application

Breakpoint

Debugged application
MalT — Performance

Testbed Specification
- Motherboard: ASUS M2V-MX_SE
- CPU: 2.2GHz AMD LE-1250
- Chipset: AMD k8 Northbridge + VIA VT 8237r Southbridge
- BIOS: Coreboot + SeaBIOS
### Table: SMM Switching and Resume (Time: \( \mu s \))

<table>
<thead>
<tr>
<th>Operations</th>
<th>Mean</th>
<th>STD</th>
<th>95% CI</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMM switching</td>
<td>3.29</td>
<td>0.08</td>
<td>[3.27, 3.32]</td>
</tr>
<tr>
<td>Command and BP checking</td>
<td>2.19</td>
<td>0.09</td>
<td>[2.15, 2.22]</td>
</tr>
<tr>
<td>Next SMI configuration</td>
<td>1.66</td>
<td>0.06</td>
<td>[1.64, 1.69]</td>
</tr>
<tr>
<td>SMM resume</td>
<td>4.58</td>
<td>0.10</td>
<td>[4.55, 4.61]</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td><strong>11.72</strong></td>
</tr>
</tbody>
</table>
MalT — Limitation

- High performance overhead on mode switch.
- Unprotected modified registers.
- Vulnerable to external timing attack.
Ninja on ARM Architecture

- Non-secure Domain
  - Rich OS
    - App
    - App
    - Malware
- Secure Domain
  - Secure Interrupt Handler
  - Trace Subsystem
  - Debug Subsystem
- Remote Debugging Client

- Secure Interrupt
- ERET
- Secure Port
Ninja on ARM Architecture

- Use TrustZone as the isolated execution environment.

- The debug subsystem is similar to MalT while the trace subsystem is immune to timing attacks.

- Modified registers are protected via hardware traps.
Ninja — Performance

- 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) [6] v1.1 and Android 5.1.1
### Table: Performance Scores Evaluated by CF-Bench [7]

<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>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>Instruction Tracing</td>
<td>6452</td>
<td>4x</td>
<td>122</td>
</tr>
</tbody>
</table>
Table: Time consumption of domain switching (Time: μs)

<table>
<thead>
<tr>
<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>
</tr>
<tr>
<td>✓</td>
<td>✗</td>
<td>0.202</td>
<td>0.013</td>
<td>[0.197, 0.207]</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>0.342</td>
<td>0.021</td>
<td>[0.334, 0.349]</td>
</tr>
</tbody>
</table>
Ninja — Limitation

- OS-related tracing requires software-based approach to fill semantic gaps, which involves performance overhead.

- Malware may intentionally enable the ETM or PMU to detect the analysis system.

- Hardware traps can only protect the system instruction access to the registers.
Outline

- Introduction
- Background
- Towards Transparent Malware Analysis
  - MalT on x86 Architecture
  - Ninja on ARM Architecture
- Conclusions
Conclusions

- We present MalT and Ninja, malware analysis systems in x86 and ARM architectures aiming for higher transparency.

- We consider the hardware-based approach provides better transparency than software-based approaches.

- To build a fully transparent malware analysis system, we are seeking for more hardware support.
Related Papers


References I

[1] Intel, “64 and IA-32 architectures software developer’s manual: Volume 3C,”


Thank you!

Questions?

zhenyu.ning@wayne.edu & fengwei@wayne.edu

http://compass.cs.wayne.edu
Hardware Traps

Non-secure Domain

```
......
MRS X0, PMCR_EL0
MOV X1, #1
AND X0, X0, X1
 ......
```
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
Hardware Traps

Non-secure Domain

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

Secure Domain

Analyzing the instruction
MOV X0, #0x41013000
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

---

44
Hardware Traps

Non-secure Domain

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

Secure Domain

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

MDCR_EL3.TPM=1