Transparent Malware Debugging on x86 and ARM

Zhenyu Ning and Fengwei Zhang

COMPASS Lab
Wayne State University

April 27, 2018

Outline

WAYNE STATE UNIVERSITY

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

Outline

WAYNE STATE UNIVERSITY

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

What is transparent malware analysis?

WAYNE STATE UNIVERSITY

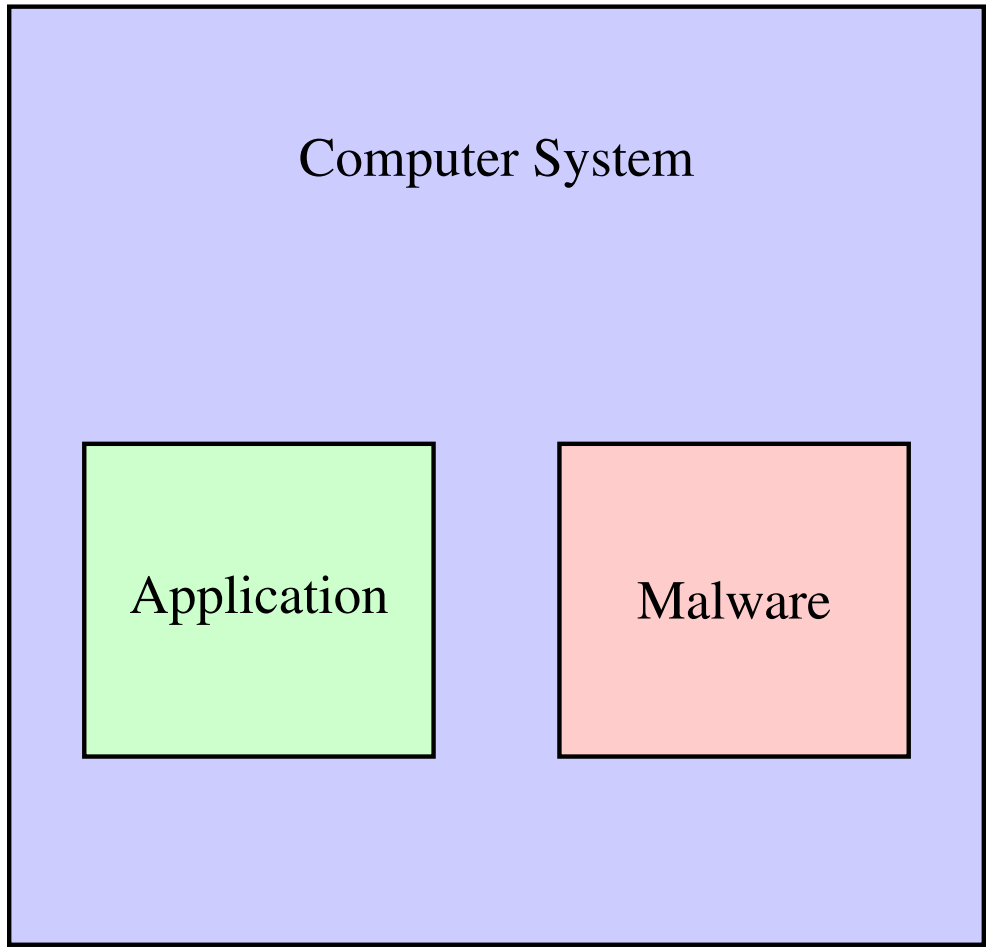
What is transparent malware analysis?

- Analyzing the malware without being aware.
- "Transparent" means that the malware cannot detect it.

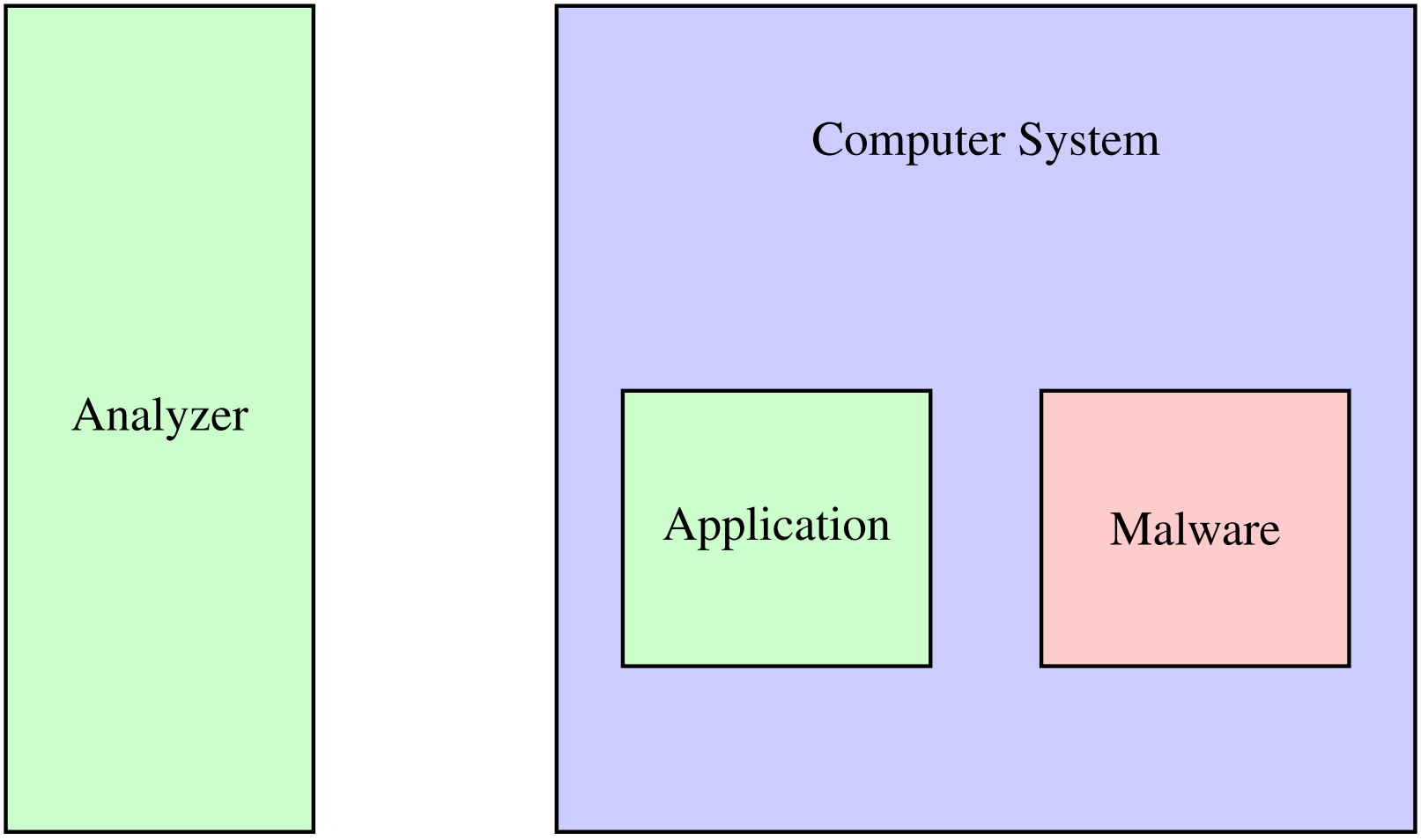
WAYNE STATE UNIVERSITY

Why transparency is important?

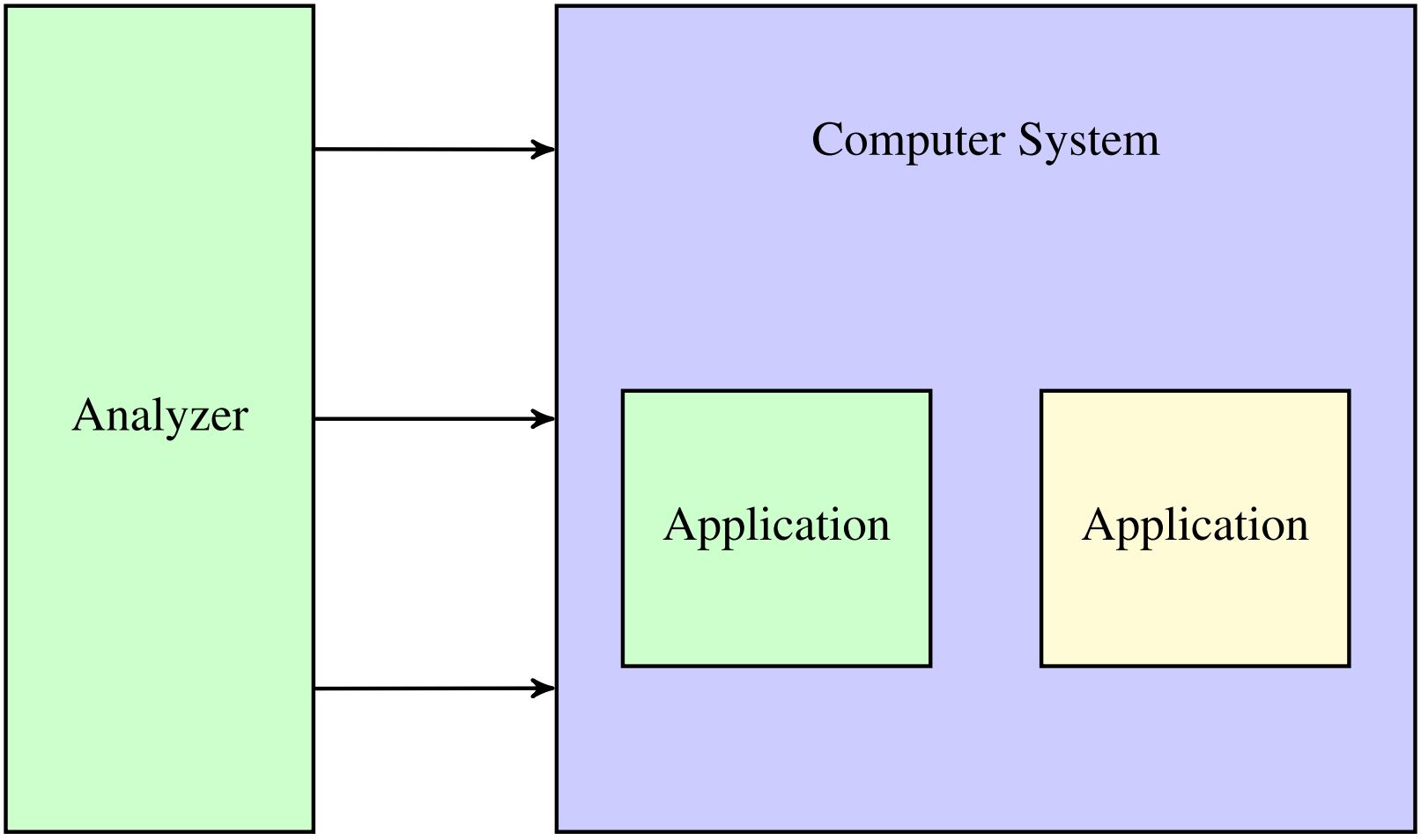
Evasive Malware



Evasive Malware

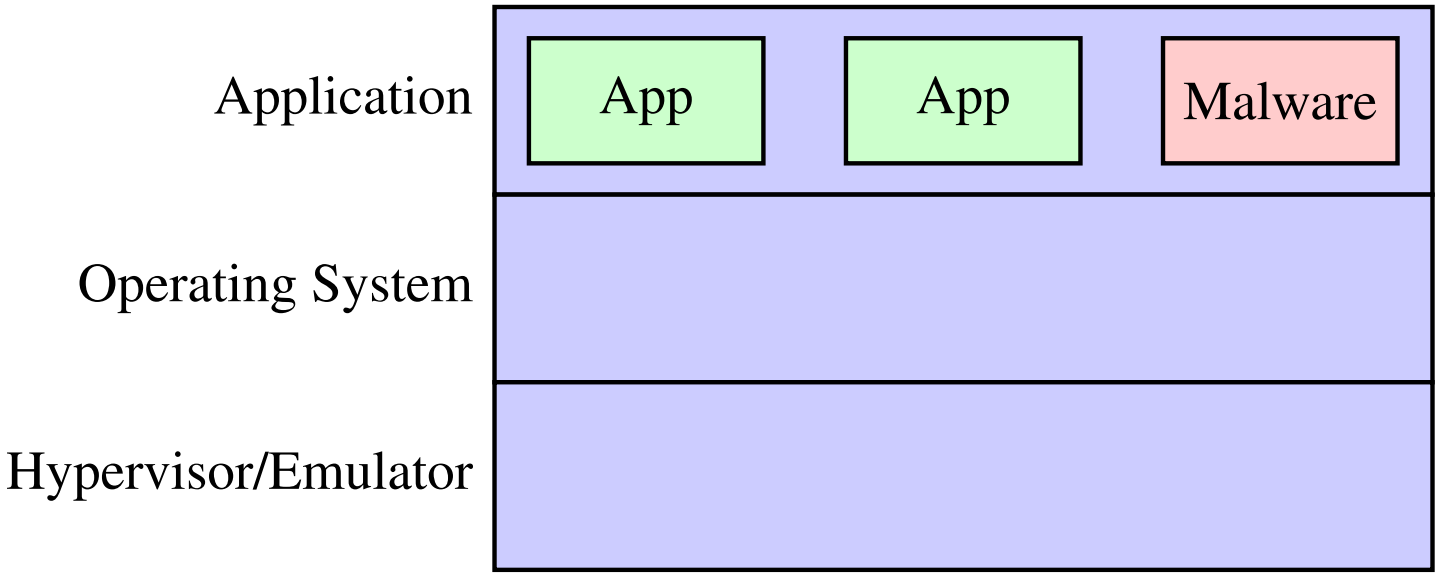


Evasive Malware

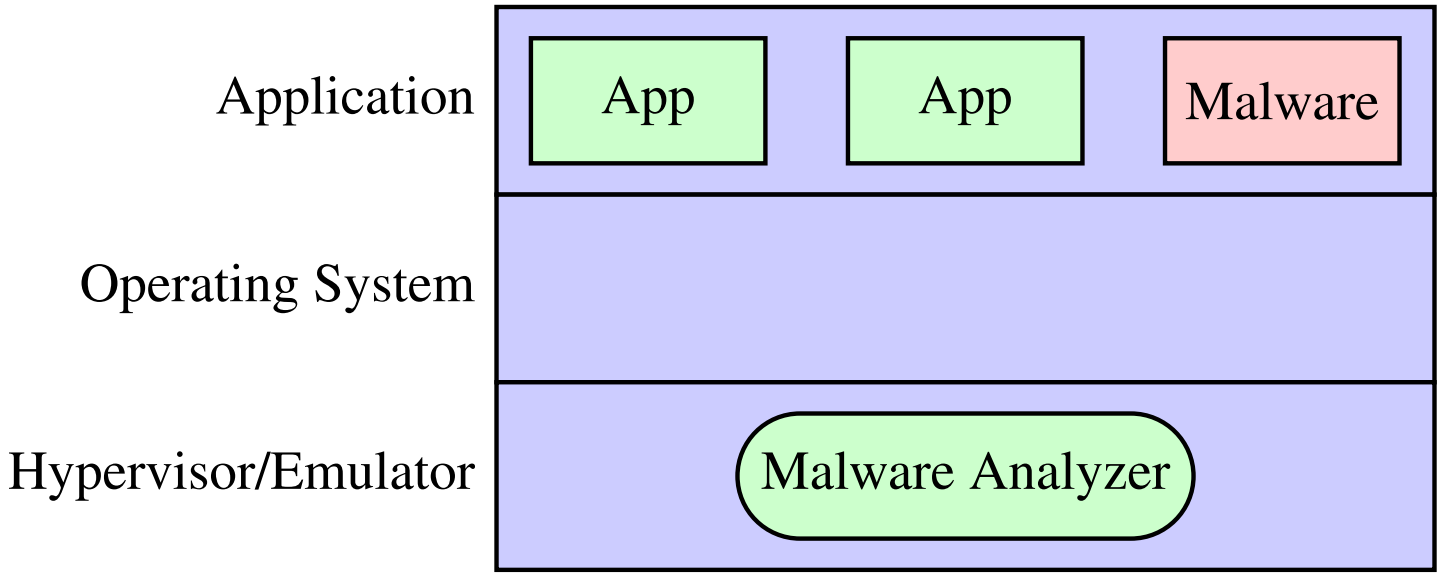


What is the current state of malware analysis systems?

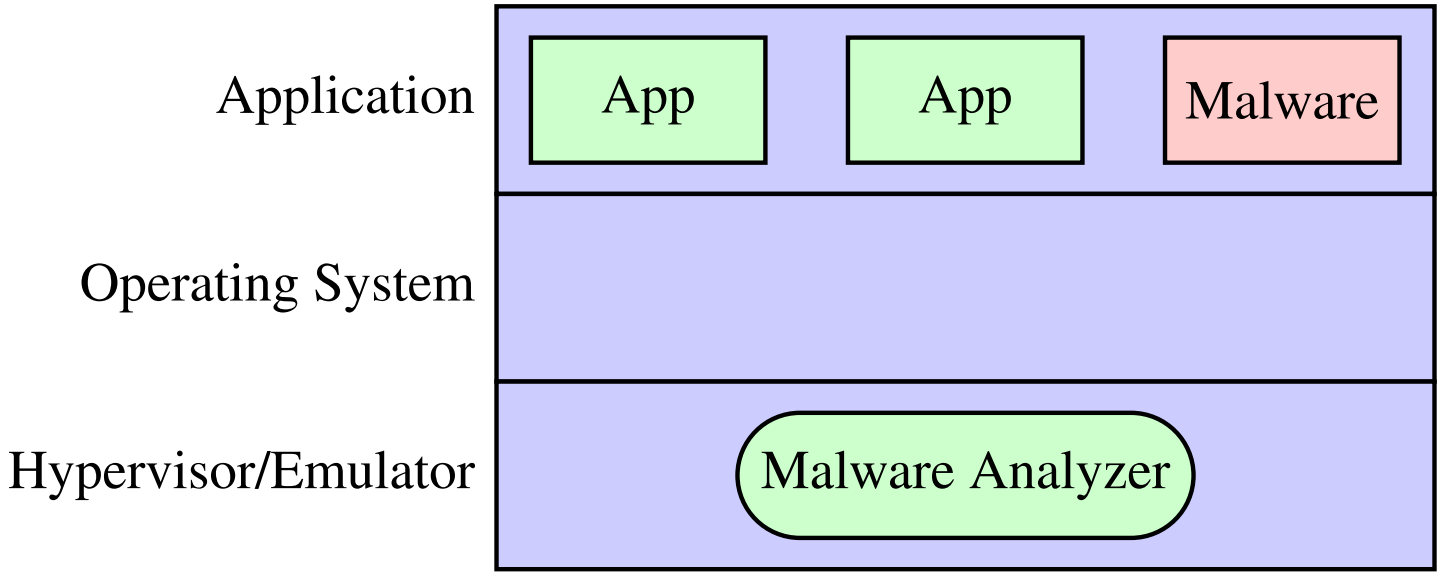
Malware Analysis



Malware Analysis

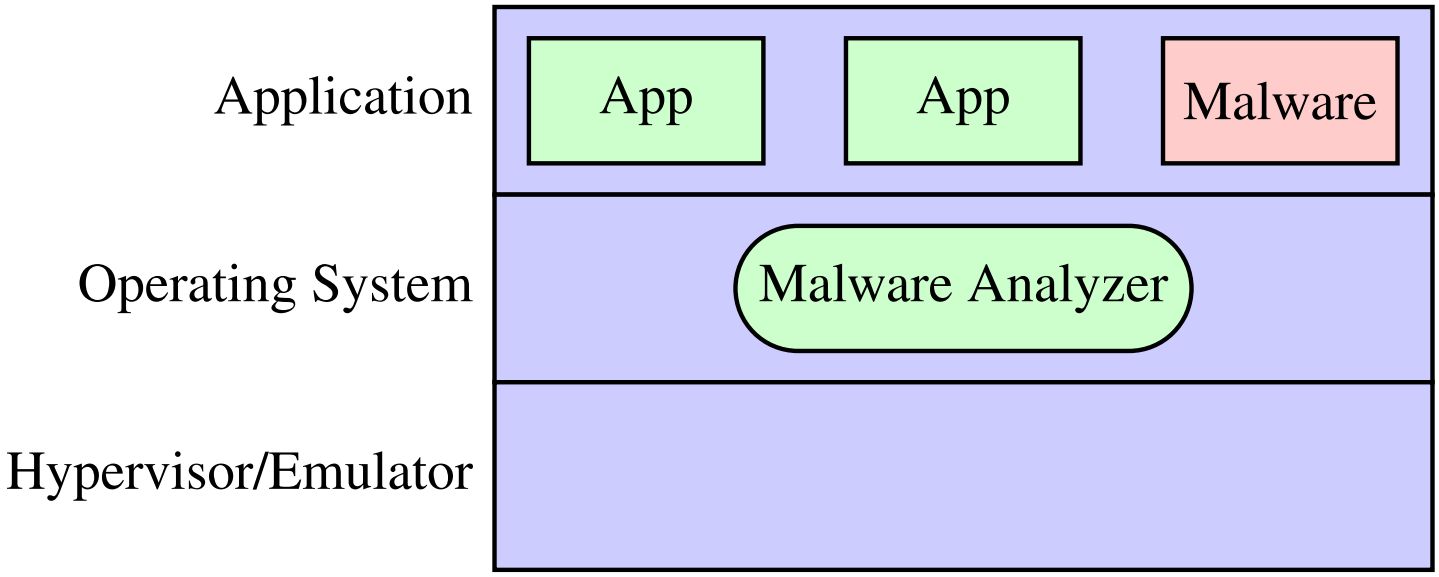


Malware Analysis

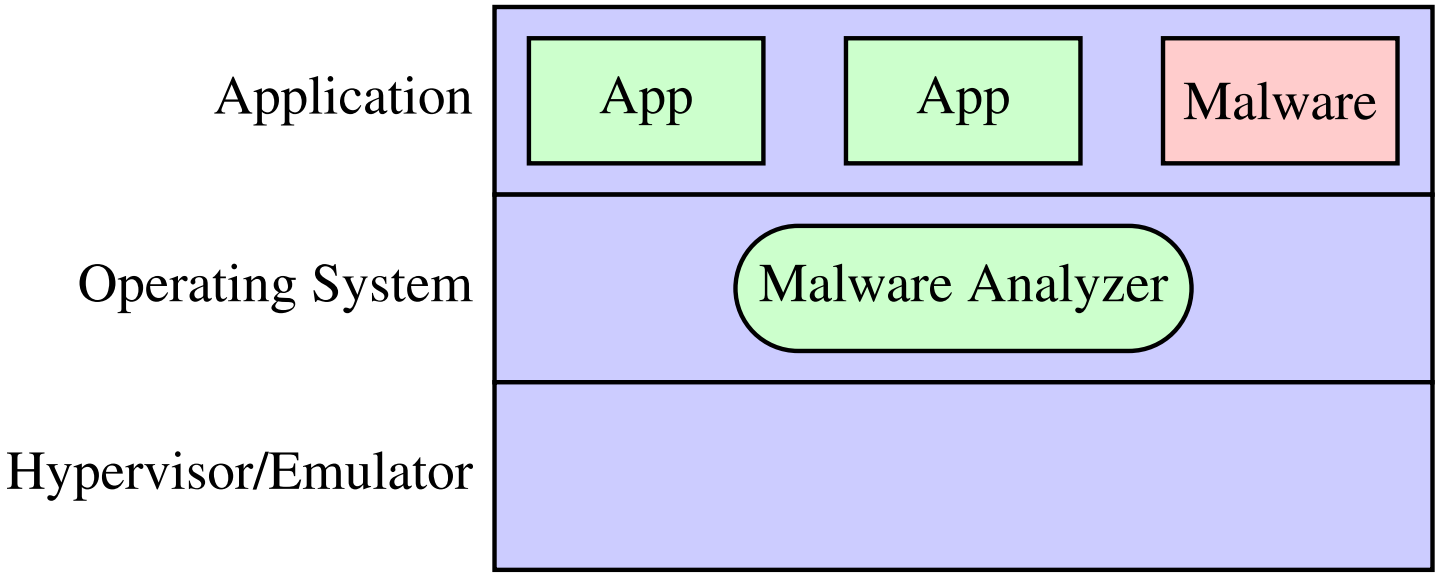


- Unarmed to anti-virtualization or anti-emulation techniques.
- Large performance overhead.

Malware Analysis



Malware Analysis



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

WAYNE STATE UNIVERSITY

What makes a transparent malware analysis system?

Transparency Requirements

WAYNE STATE UNIVERSITY

- 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

WAYNE STATE UNIVERSITY

- 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

WAYNE STATE UNIVERSITY

- 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

WAYNE STATE UNIVERSITY

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

System Management Mode

WAYNE STATE UNIVERSITY

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

WAYNE STATE UNIVERSITY

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

WAYNE STATE UNIVERSITY

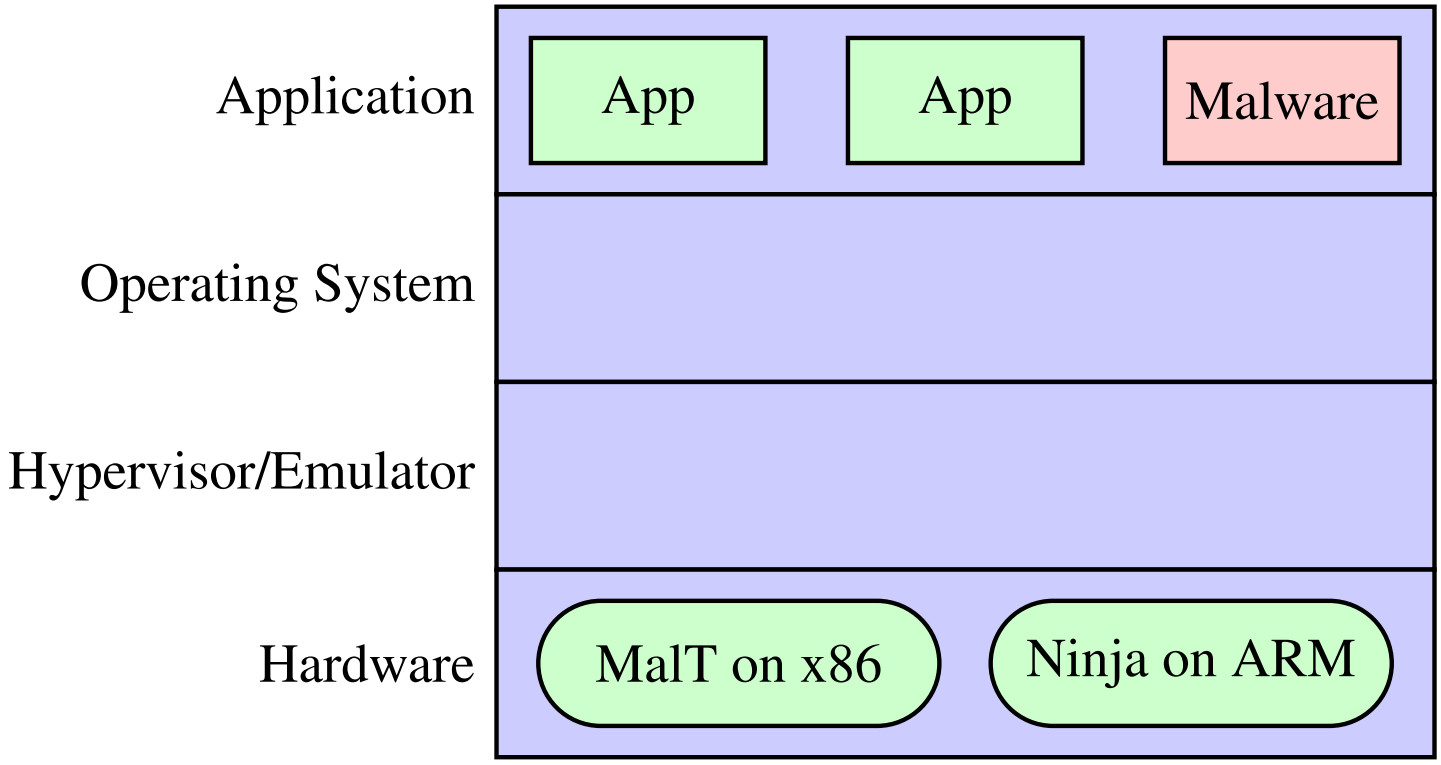
- The Performance Monitor Unit (PMU) [3, 4] leverages a set of performance counter registers to count the occurrence of different CPU events.
- The Embedded Trace Macrocell (ETM) [5] traces the instructions and data of the system, and output the trace stream into pre-allocated buffers on the chip.
- The PMU exists in both x86 and ARM architecture while the ETM is ARM special hardware.

Outline

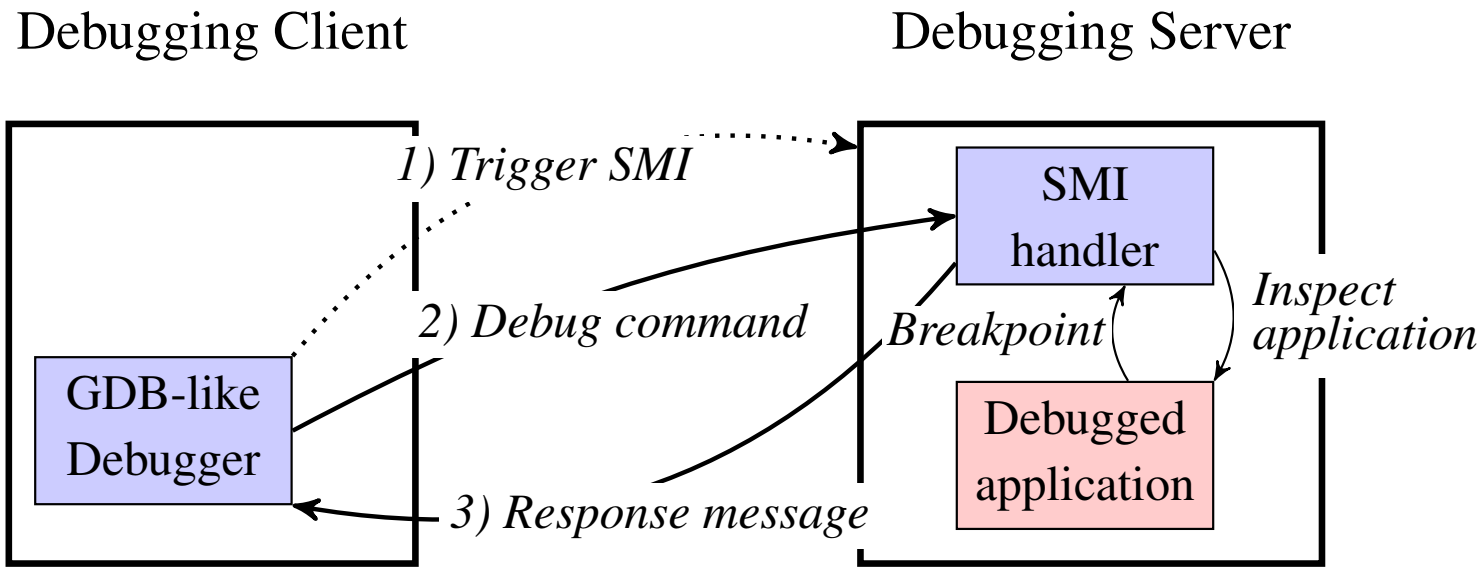
WAYNE STATE UNIVERSITY

- 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



MalT on x86 Architecture



MalT — Performance

WAYNE STATE UNIVERSITY

- Testbed Specification
  - Motherboard: ASUS M2V-MX_SE
  - CPU: 2.2GHz AMD LE-1250
  - Chipset: AMD k8 Northbridge + VIA VT 8237r Southbridge
  - BIOS: Coreboot + SeaBIOS

MalT — Performance

WAYNE STATE UNIVERSITY

Table: SMM Switching and Resume (Time: μs)

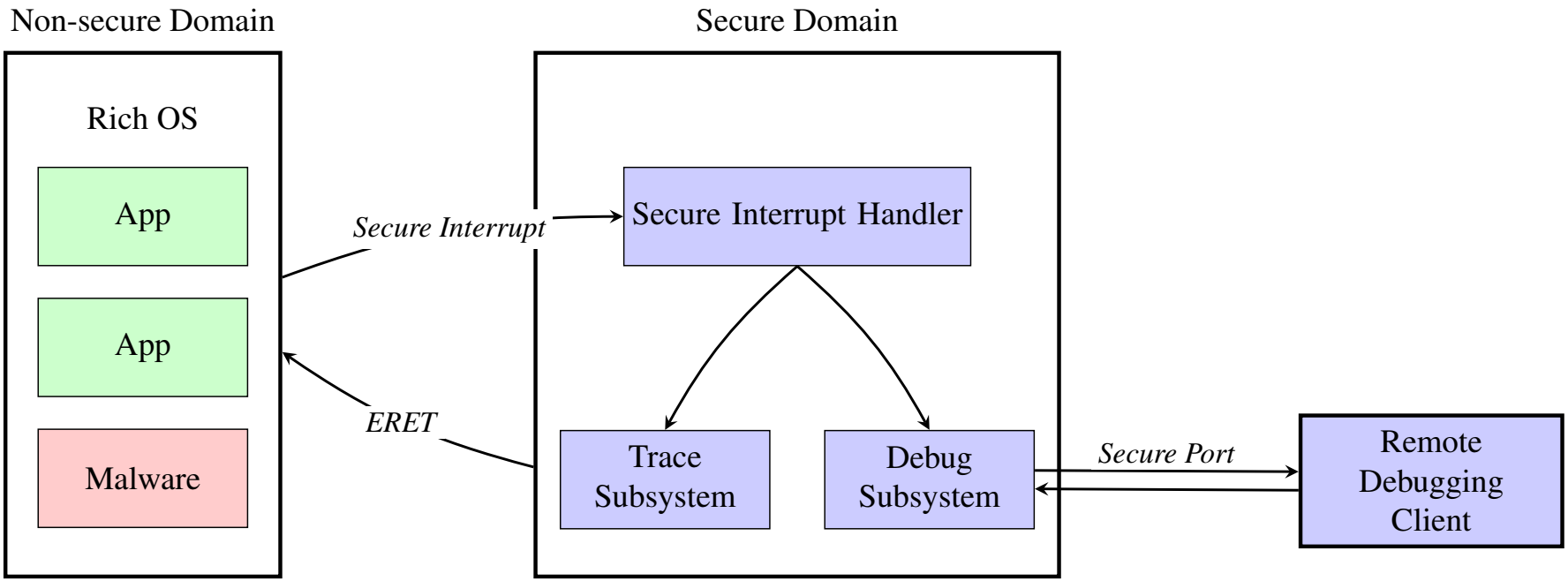
Operations	**Mean**	**STD**	**95% CI**
SMM switching	3.29	0.08	[3.27, 3.32]
Command and BP checking	2.19	0.09	[2.15, 2.22]
Next SMI configuration	1.66	0.06	[1.64, 1.69]
SMM resume	4.58	0.10	[4.55, 4.61]
Total	11.72		

MalT — Limitation

WAYNE STATE UNIVERSITY

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

Ninja on ARM Architecture



Ninja on ARM Architecture

WAYNE STATE UNIVERSITY

- 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

WAYNE STATE UNIVERSITY

- 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

Ninja — Performance

WAYNE STATE UNIVERSITY

Table: Performance Scores Evaluated by CF-Bench [7]

	Native Scores		**Java Scores**		**Overall Scores**	
	Mean	Slowdown	Mean	Slowdown	Mean	Slowdown
Tracing Disabled	25380		18758		21407	
Instruction Tracing	25364	1x	18673	1x	21349	1x
System call Tracing	25360	1x	18664	1x	21342	1x
Instruction Tracing	6452	4x	122	154x	2654	8x

Ninja — Performance

WAYNE STATE UNIVERSITY

Table: Time consumption of domain switching (Time: μs)

ATF Enabled	Ninja Enabled	Mean	STD	95% CI
×	×	0.007	0.000	[0.007, 0.007]
✓	×	0.202	0.013	[0.197, 0.207]
✓	✓	0.342	0.021	[0.334, 0.349]

Ninja — Limitation

WAYNE STATE UNIVERSITY

- 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

WAYNE STATE UNIVERSITY

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

Conclusions

WAYNE STATE UNIVERSITY

- 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

WAYNE STATE UNIVERSITY

USENIX Security'17 — Zhenyu Ning and Fengwei Zhang. Ninja: Towards Transparent Tracing and Debugging on ARM. In *Proceedings of The 26th USENIX Security Symposium*, Vancouver, BC, Canada, August 2017.

S&P'15 — Fengwei Zhang, Kevin Leach, Angelos Stavrou, and Haining Wang. Using Hardware Features for Increased Debugging Transparency. In *Proceedings of The 36th IEEE Symposium on Security and Privacy*, San Jose, CA, May 2015.

References I

[1] Intel, "64 and IA-32 architectures software developer's manual: Volume 3C," https://software.intel.com/sites/default/files/managed/a4/60/325384-sdm-vol-3abcd.pdf.

[2] ARM Ltd., "TrustZone Security Whitepaper," http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.prd29-genc-009492c/index.html.

[3] ——, "ARMv8-A Reference Manual," http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0487a.k/index.html.

[4] Intel, "64 and IA-32 architectures software developer's manual: Volume 3B," https://software.intel.com/sites/default/files/managed/a4/60/325384-sdm-vol-3abcd.pdf.

[5] ARM Ltd., "Embedded Trace Macrocell Architecture Specification," http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ihi0014q/index.html.

[6] ——, "ARM Trusted Firmware," https://github.com/ARM-software/arm-trusted-firmware.

[7] Chainfire, "CF-Bench," https://play.google.com/store/apps/details?id=eu.chainfire.cfbench.

Thank you!

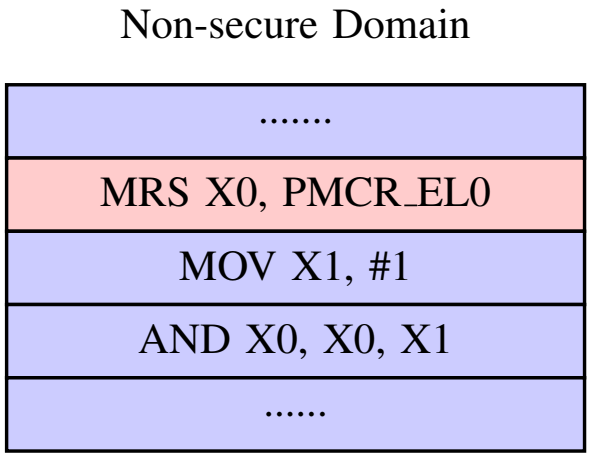
WAYNE STATE UNIVERSITY

Questions?

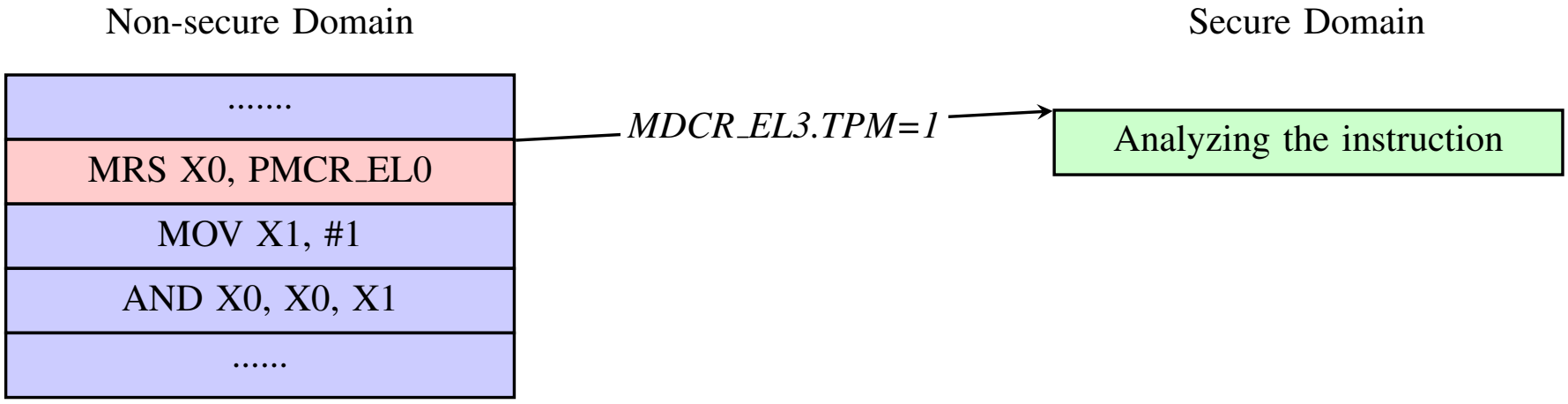
zhenyu.ning@wayne.edu & fengwei@wayne.edu

`http://compass.cs.wayne.edu`

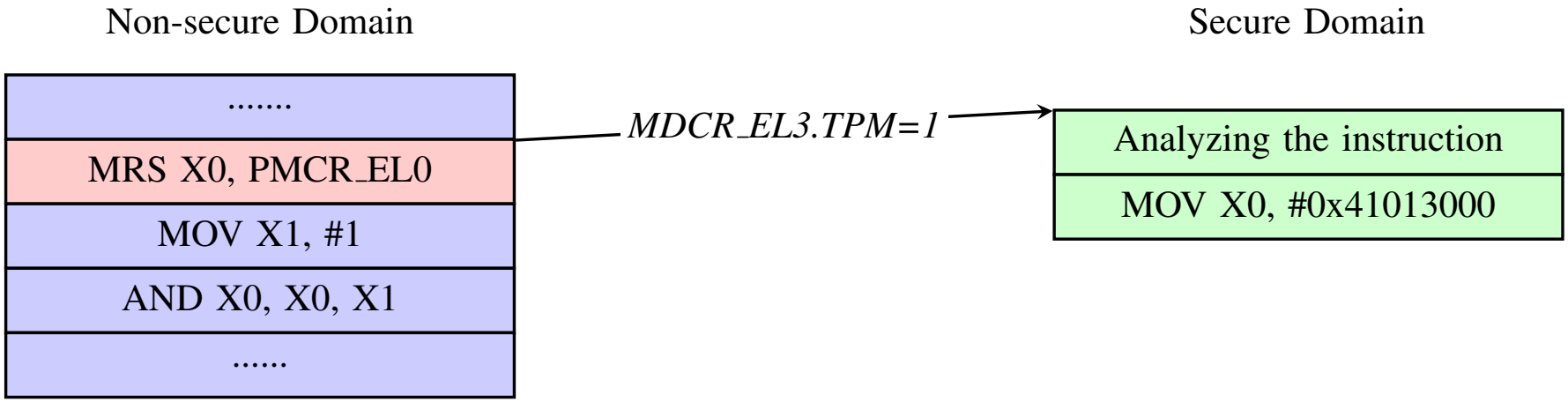
Hardware Traps



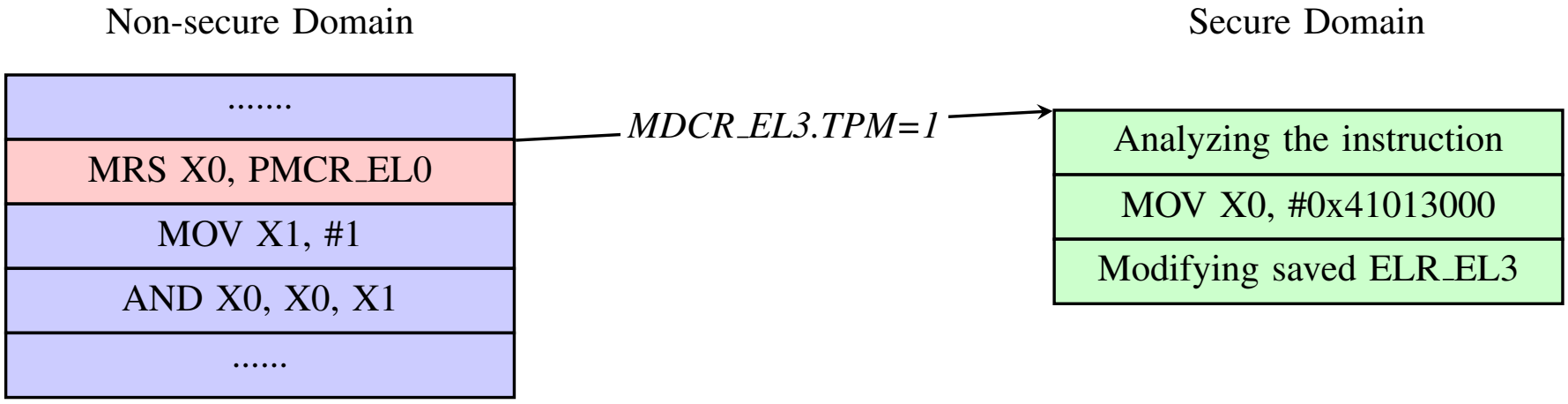
Hardware Traps



Hardware Traps



Hardware Traps



Hardware Traps

