Ninja: Towards Transparent Tracing and Debugging on ARM

Zhenyu Ning & Fengwei Zhang
Wayne State University

Outline

- Introduction
- Background
- System Architecture Implementation
- Evaluation
- Conclusion

Outline

- Introduction
- Background
- System Architecture Implementation
- Evaluation
- Conclusion

Motivation / Goal

Existing malware analysis platforms:

- Leave detectable fingerprints (like uncommon string properties in QEMU, signatures in Android Java virtual machine, and Artifacts in Linux kernel profiles)

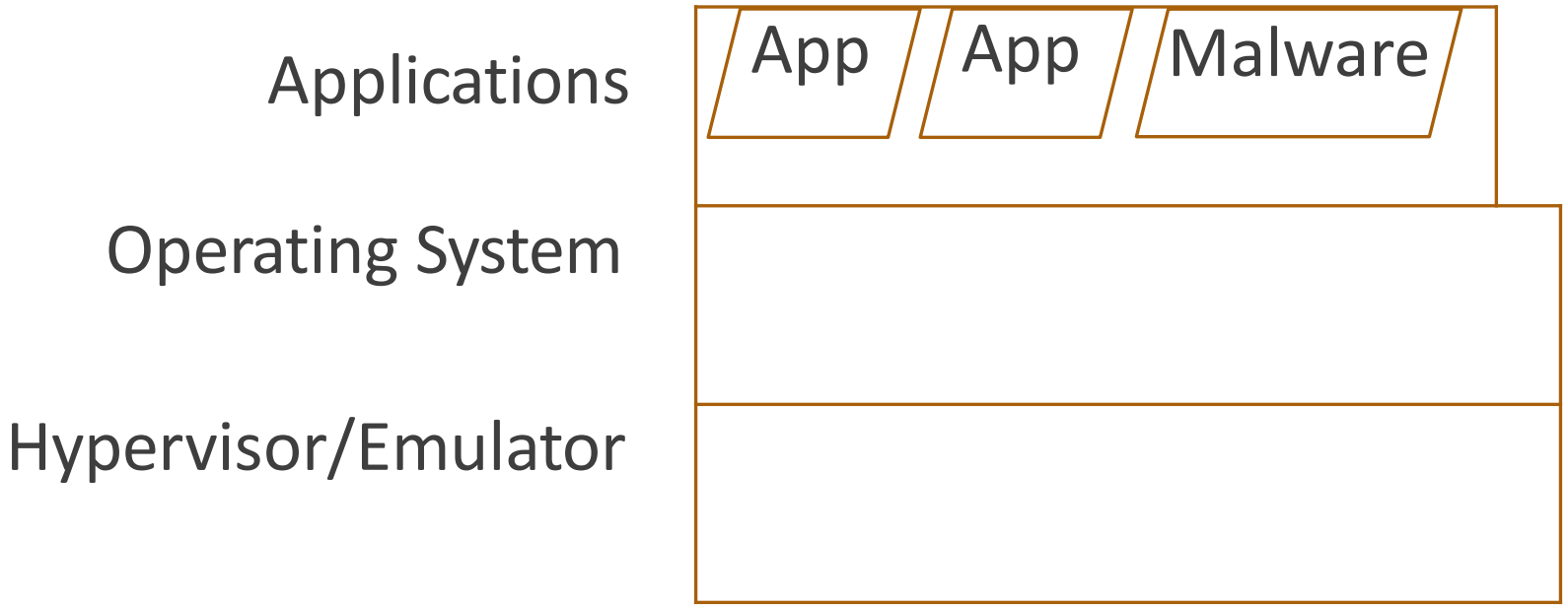
NINJA:

- Leverages a hardware-assisted isolated execution environment Trust-Zone to transparently trace and debug a target application.
- Does not modify system software and is OS-agnostic on ARM platform.

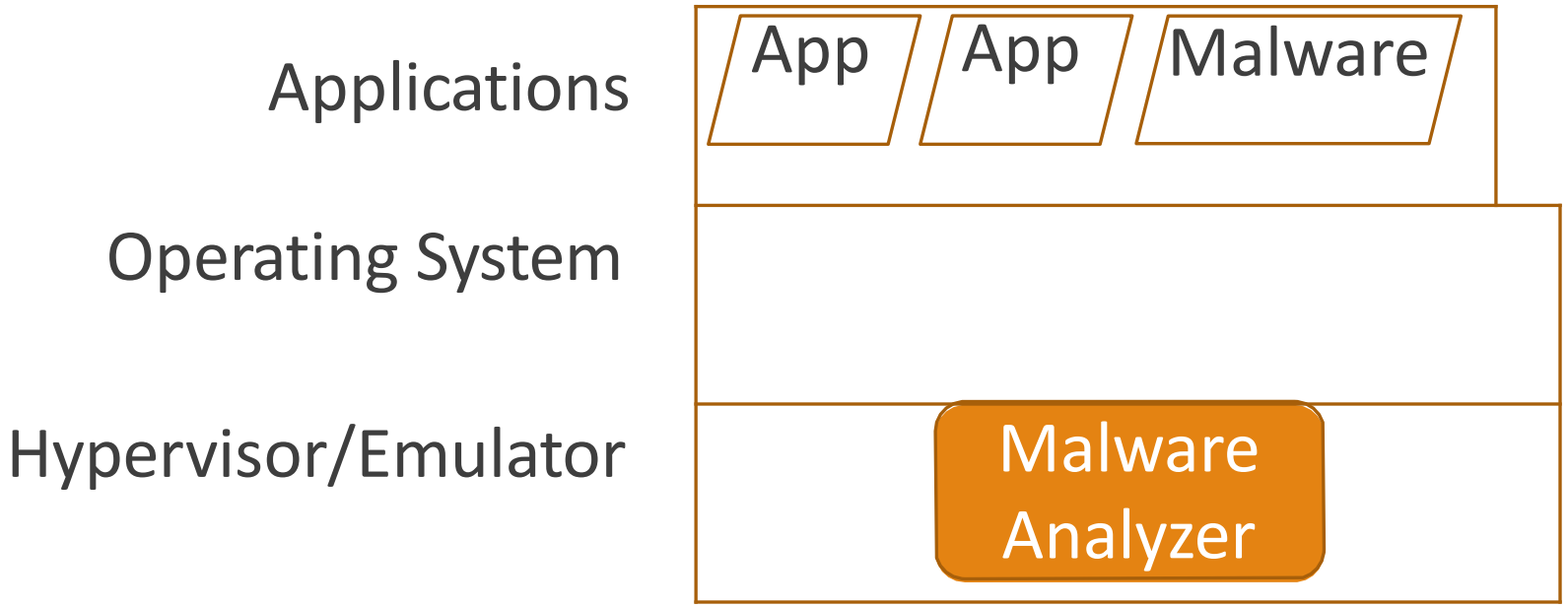
Motivation–Evasion Malware

Goal–Transparent malware analysis framework

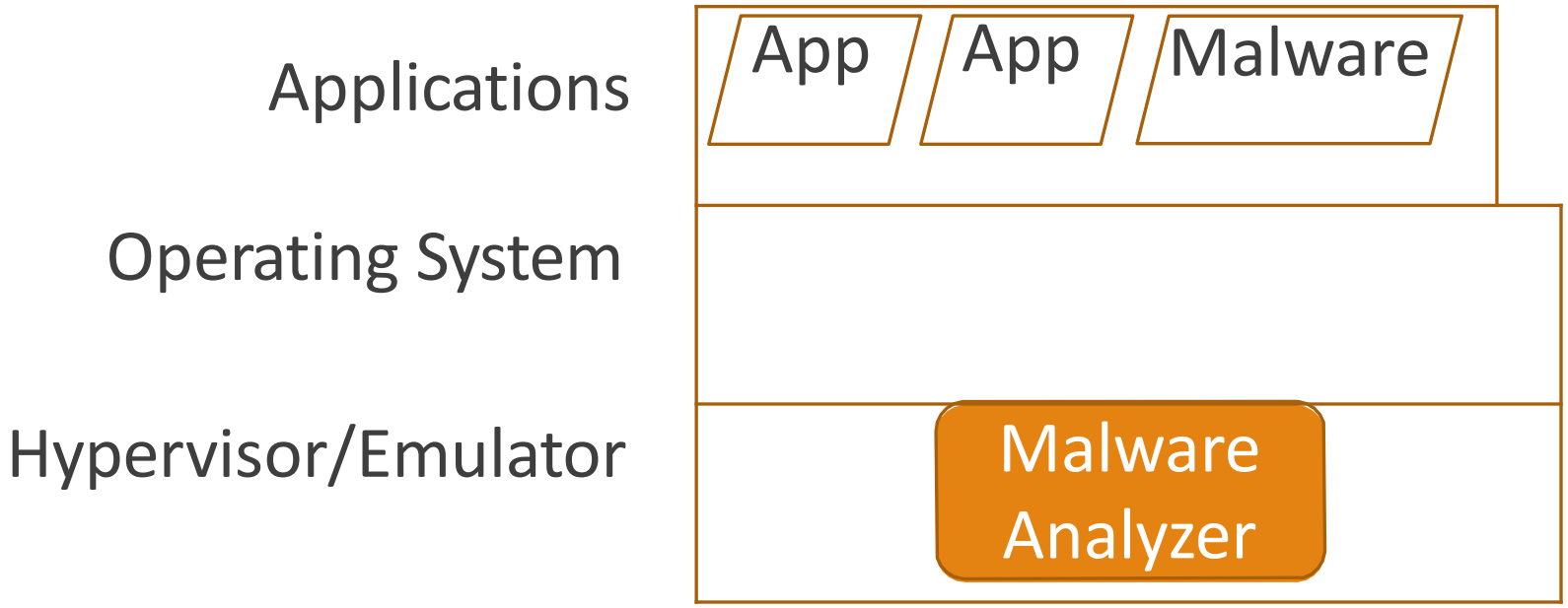
General Malware Analysis Layers



Traditional Malware Analysis



Traditional Malware Analysis

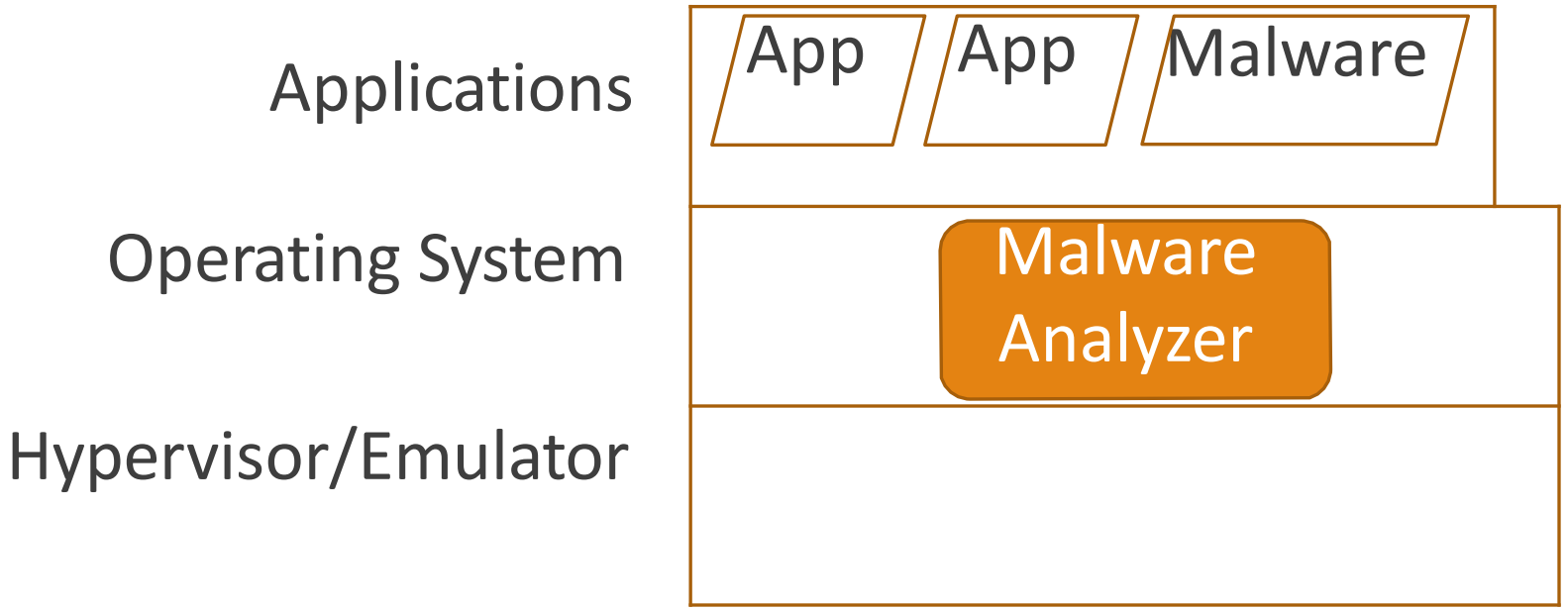


Limitation:

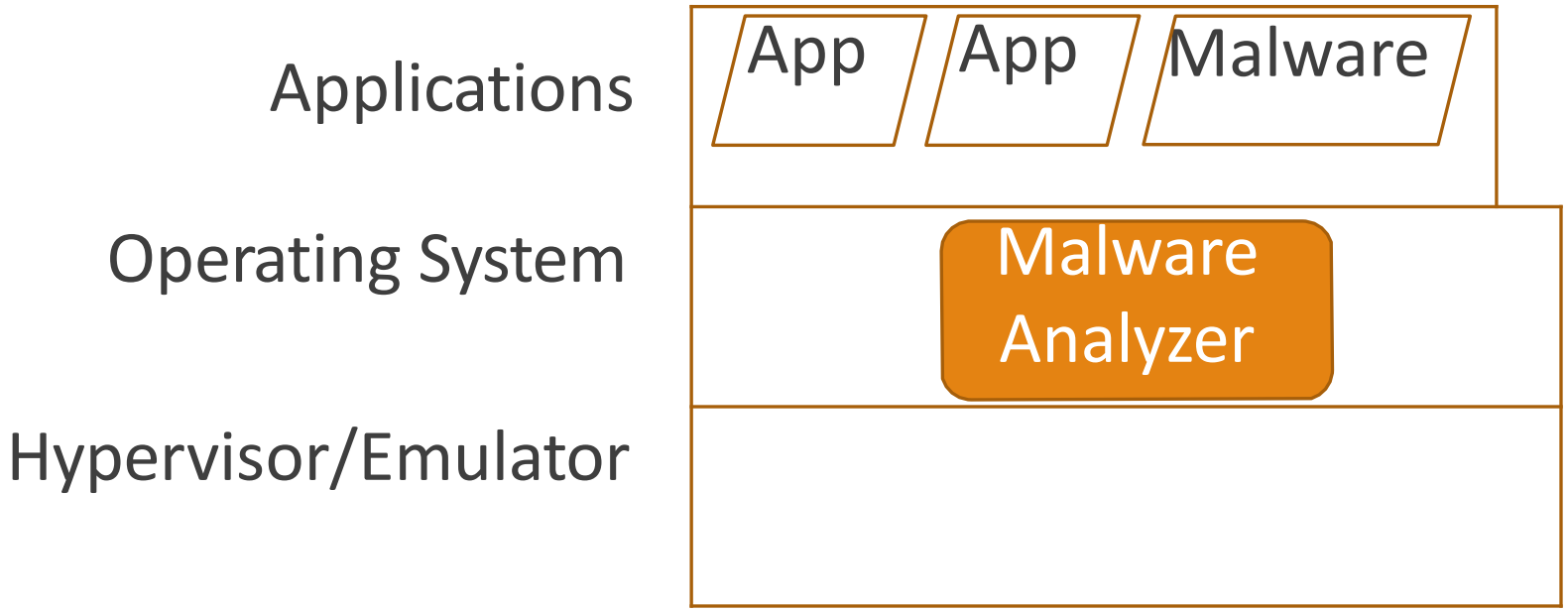
- Unarmed to anti-virtualization or anti-emulation techniques

Can be easily detected by footprints like string properties, absence of particular hardware components, and performance slowdown.

Malware Analysis



Malware Analysis

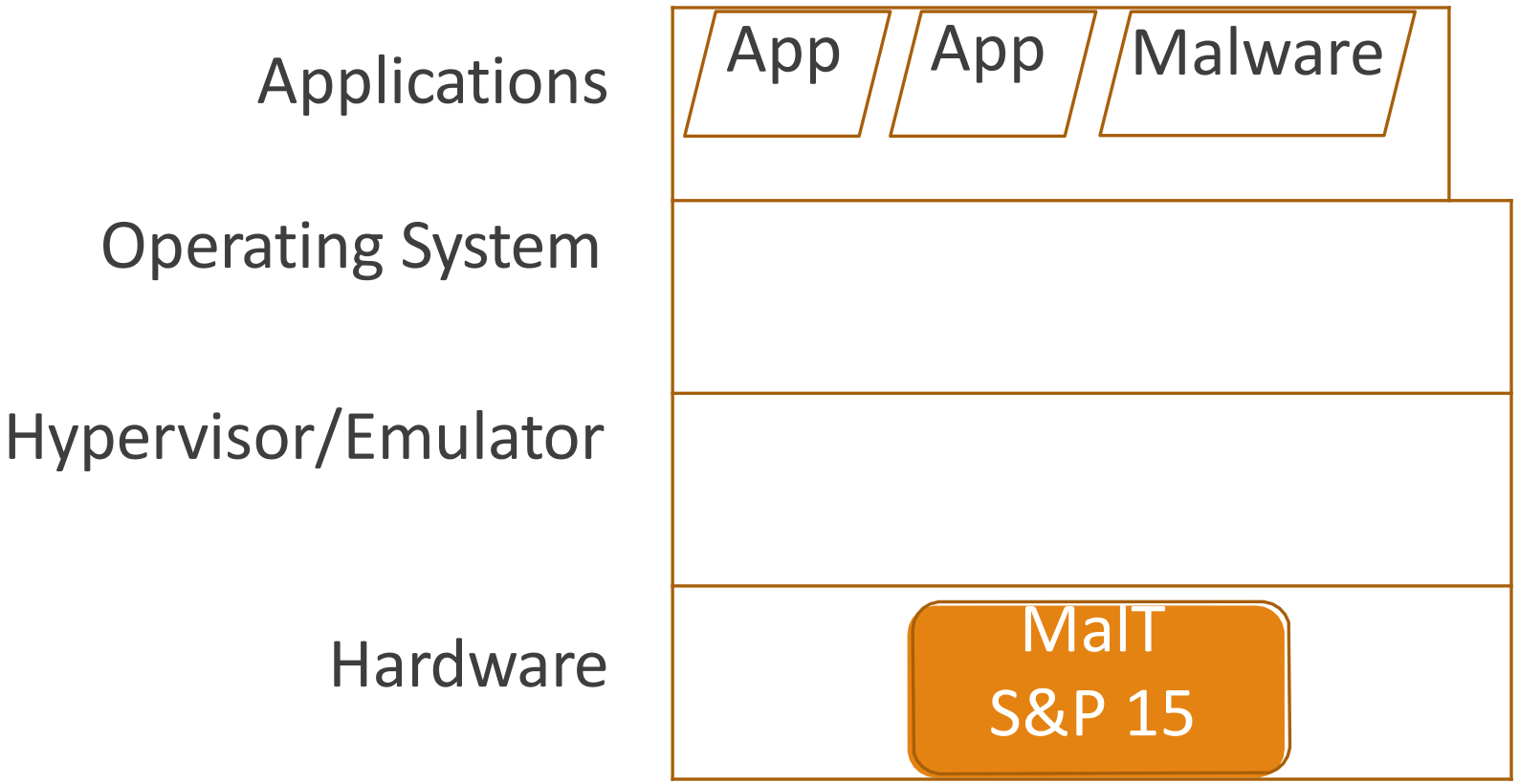


Limitation:

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

Privileged malware can even manipulate the analysis tool since they run in the same environment.

Malware Analysis



Related Work–Malt

Introduction：

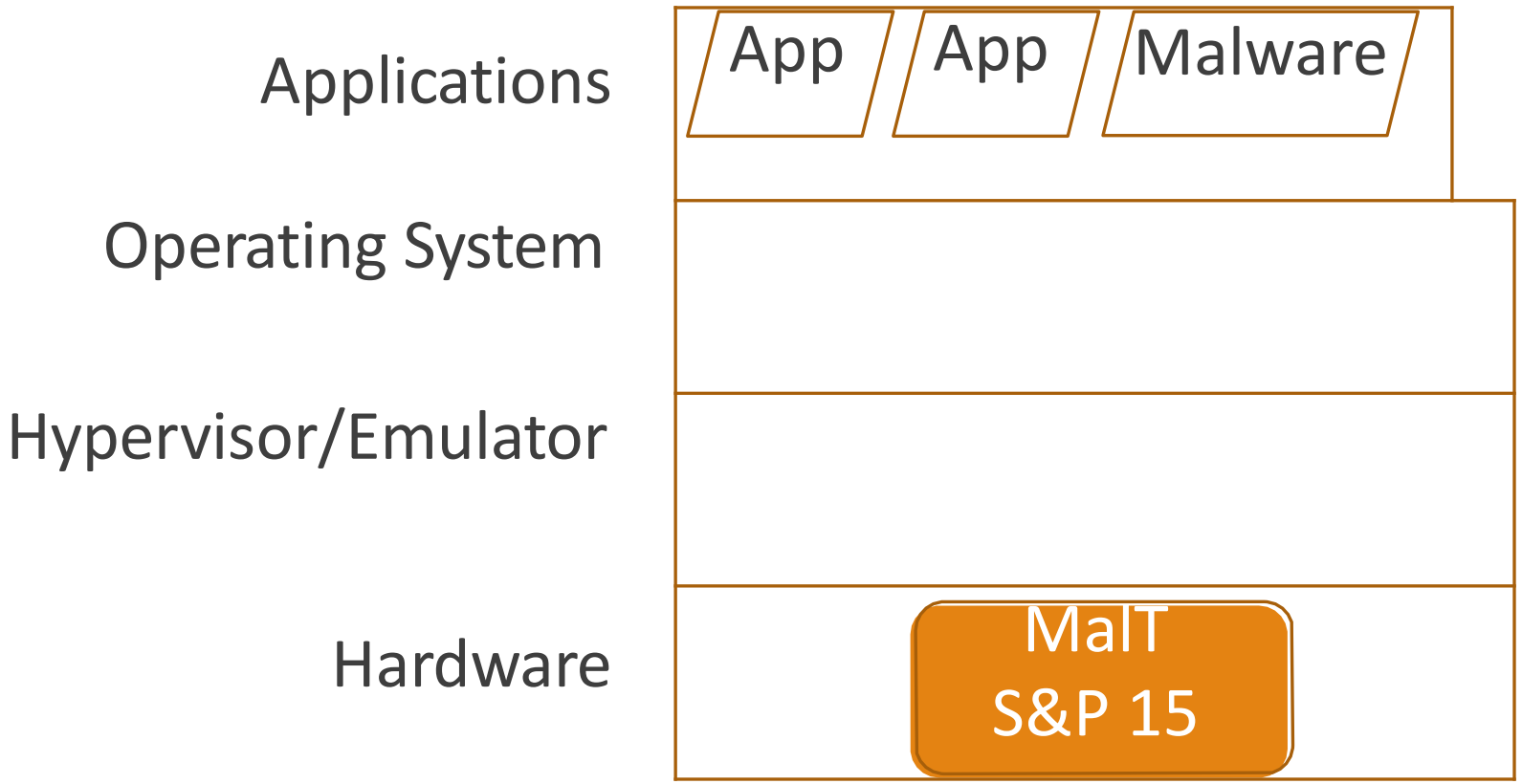
- Increases the transparency by involving System Manage Mode (SMM), special CPU mode in x86.
- Leverages Performance Monitor Unit(PMU) to monitor the program execution and switch into SMM for analysis.

Related Work–Malt

Disadvantage：

- PMU registers on MalT are accessible by privileged malware.
- SMM leads to high performance overhead.

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
  - Provide system call sequence

- An ***Analyzer*** which is responsible for the further analysis of the states
  - Record system call and send to a remote server for further analysis

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

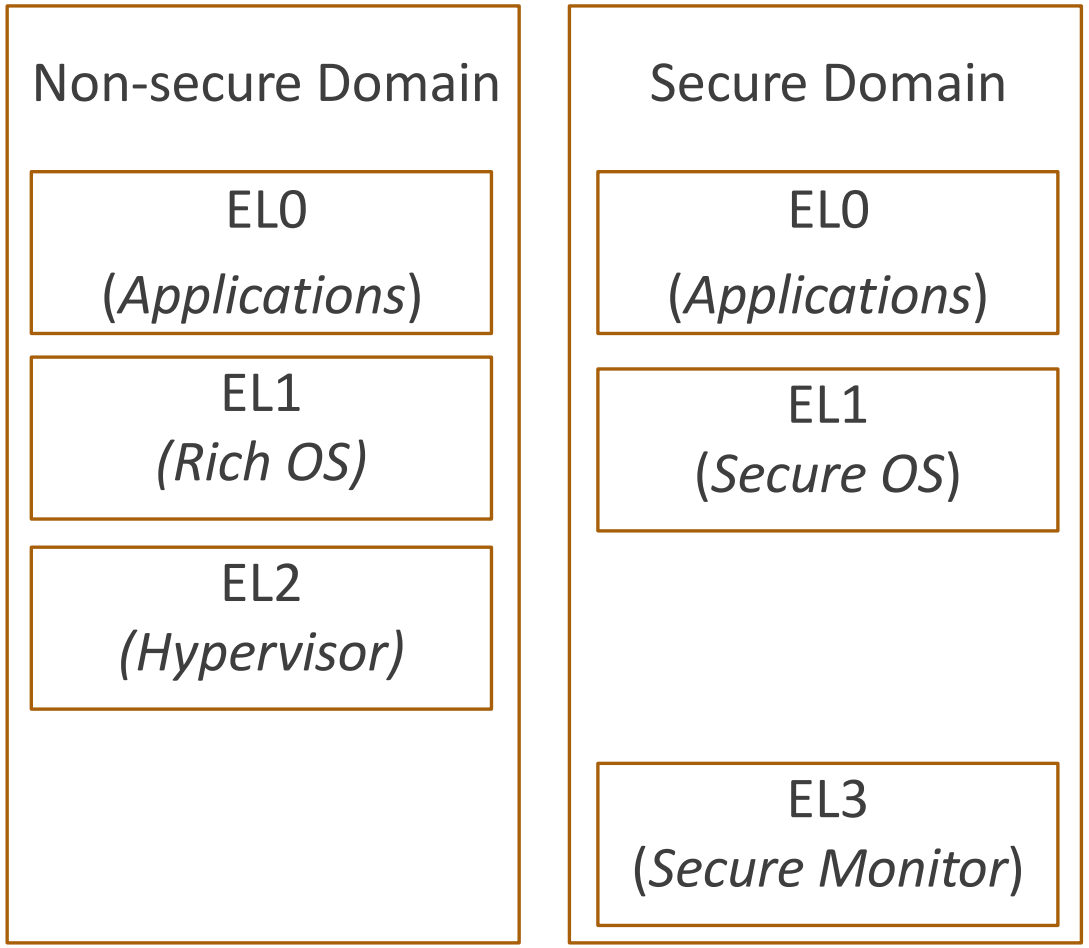
- Introduction
- Background
- System Overview
- Evaluation
- Conclusion

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.

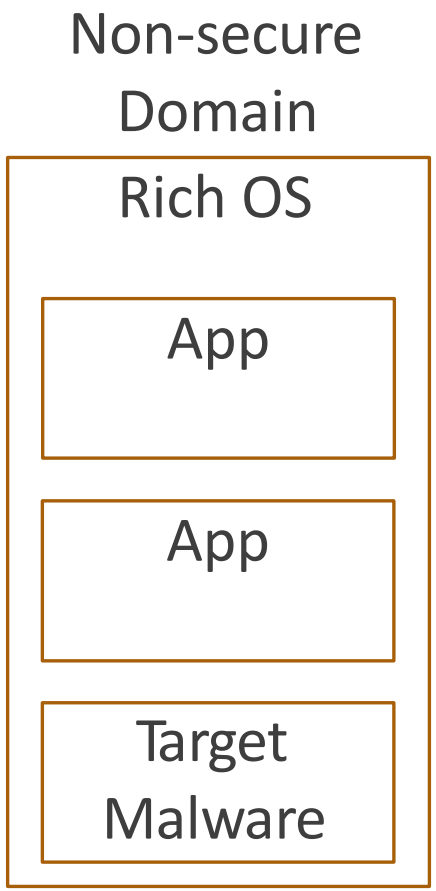
Background – PMU and ETM

- The Performance Monitor Unit (PMU) leverages a set of performance counter registers to count the occurrence of different CPU events. (用於Debug Subsystem)
- 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. (用於Trace Subsystem)
- Both PMU and ETM exist on ARM Cortex-A5x and Cortex-A7x series CPUs, and do NOT affect the performance of the CPU.

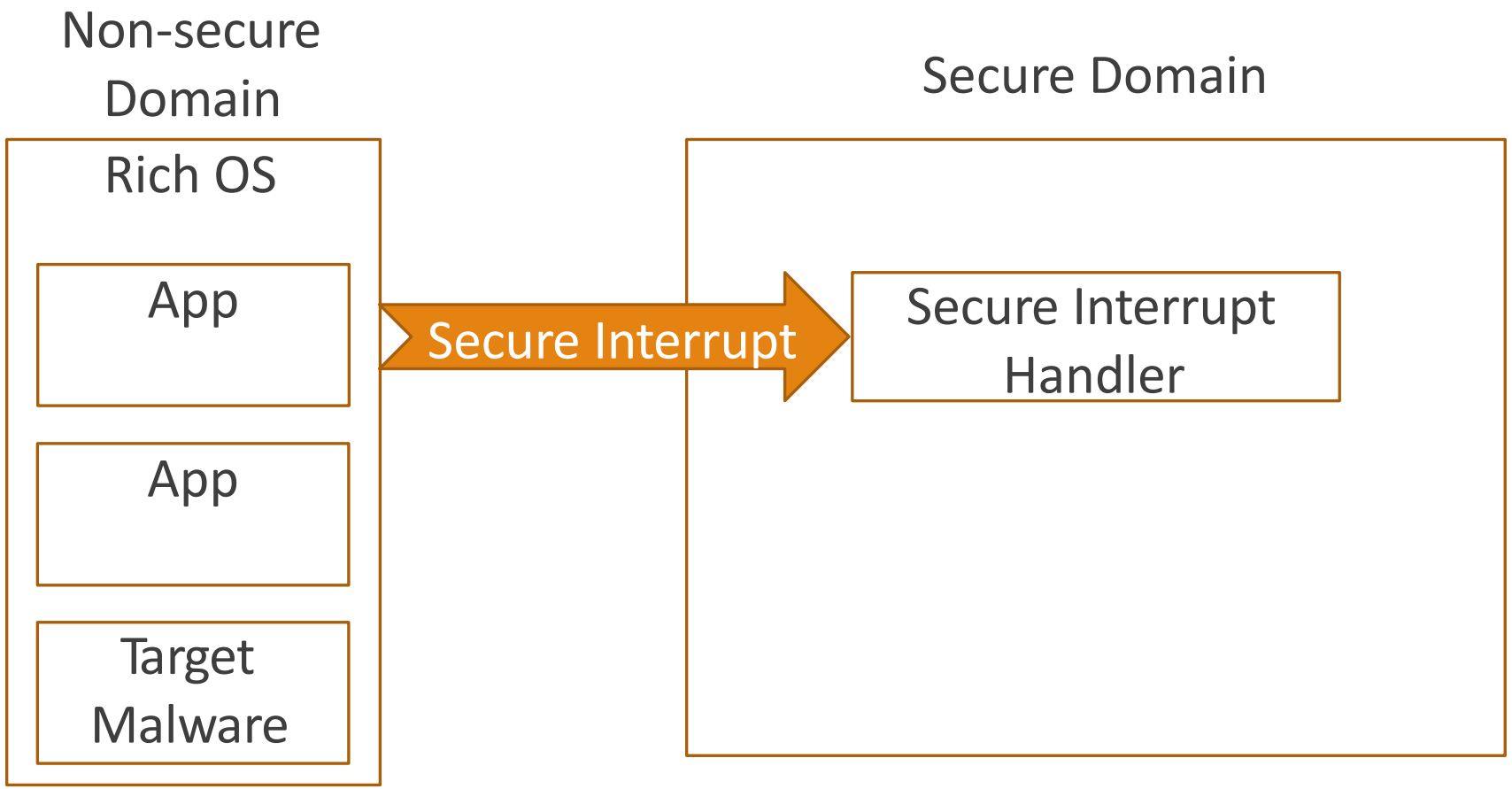
Outline

- Introduction
- Background
- System Architecture Implementation
- Evaluation
- Conclusion

Architecture

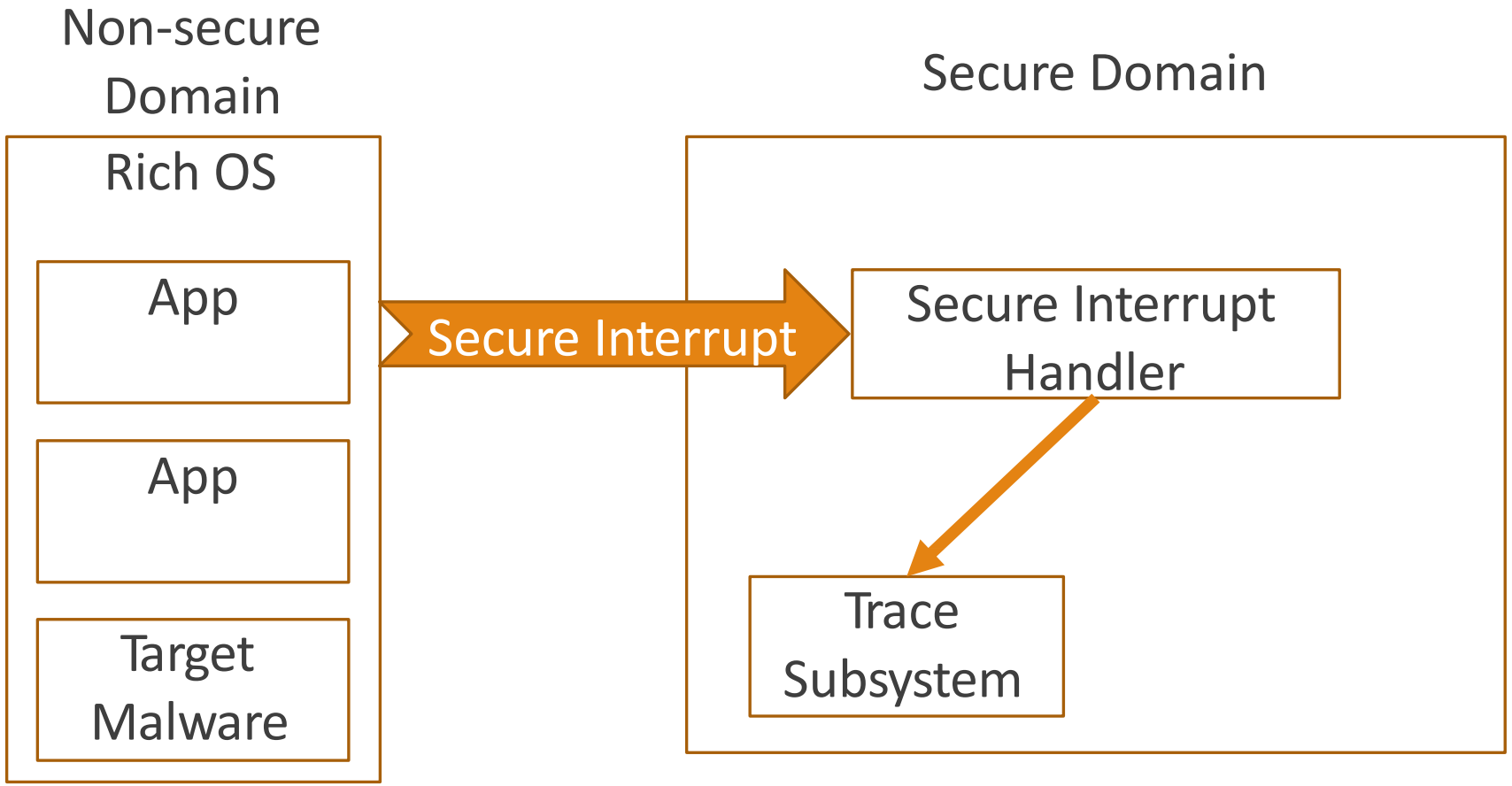


Architecture



Architecture

- TS provides the analyst the ability to trace the execution of the target application in different granularities.

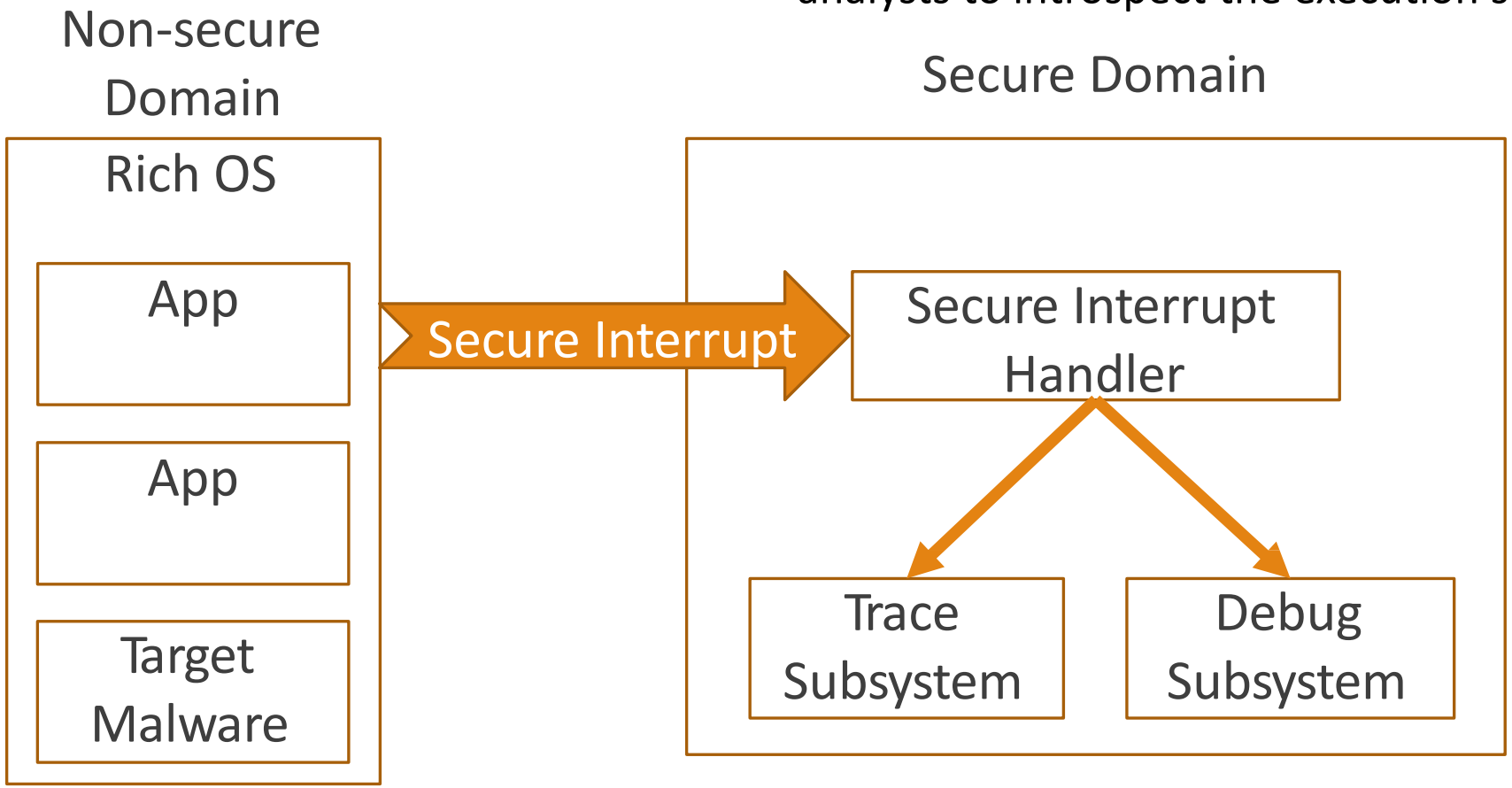


Trace Subsystem:

- Instruction Trace
- System Call Trace
- Android API Trace

Architecture

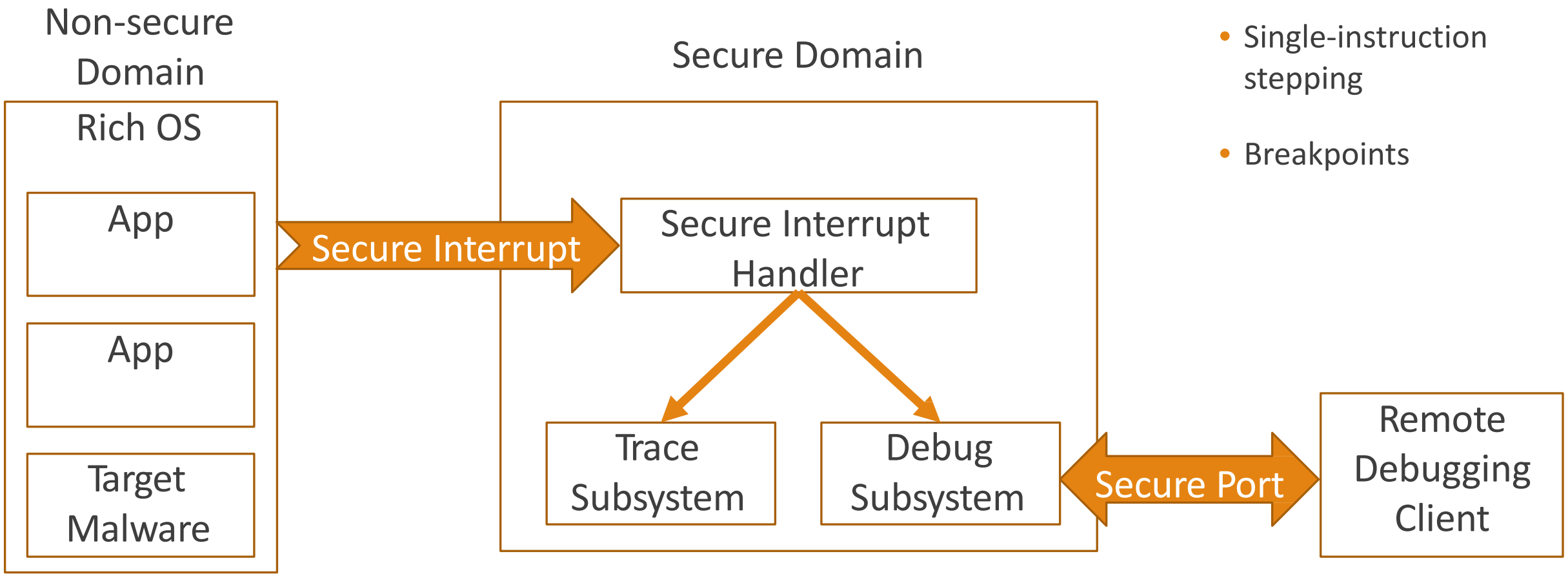
- DS is designed for manual analysis.
- Establishes a secure channel between the target executing platform and the remote debugging platform, and provides a user interface for human analysts to introspect the execution status of the target application.



Debug Subsystem:

- Single Stepping
- Breakpoints
- Memory R/W

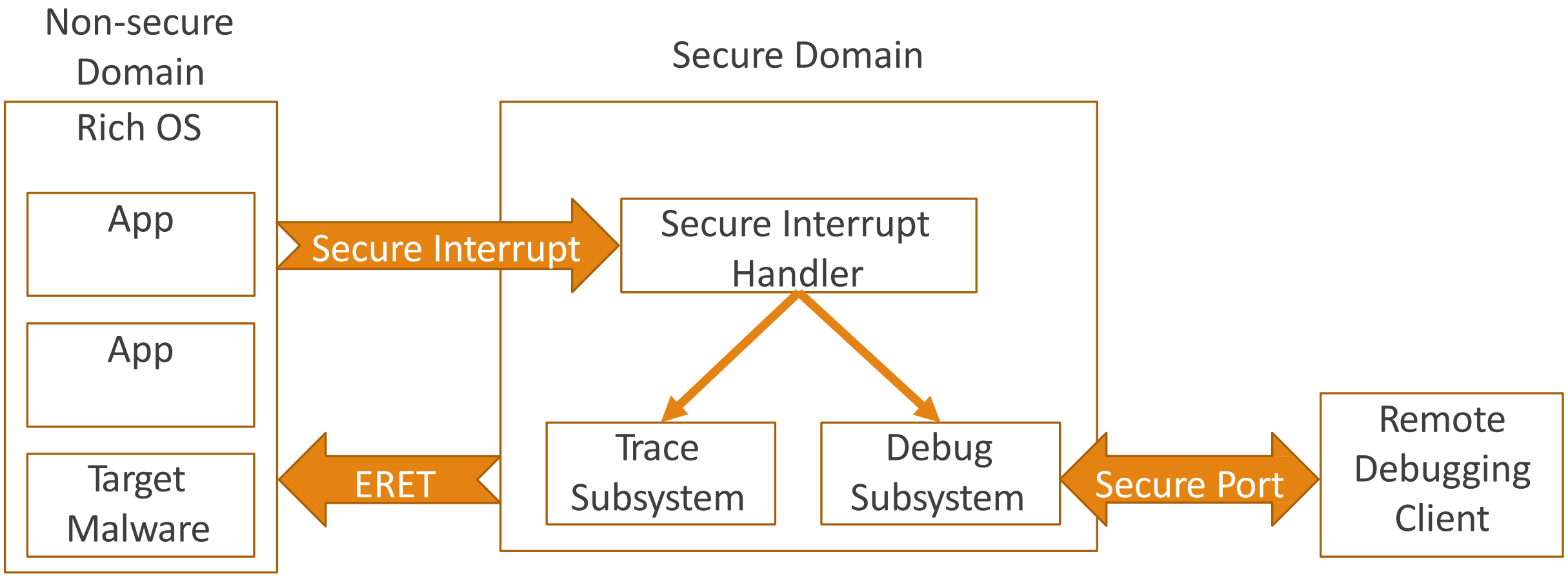
Architecture



Secure Serial Port:

- Single-instruction stepping
- Breakpoints

Architecture



System Architecture Implementation

Protect Modified Registers

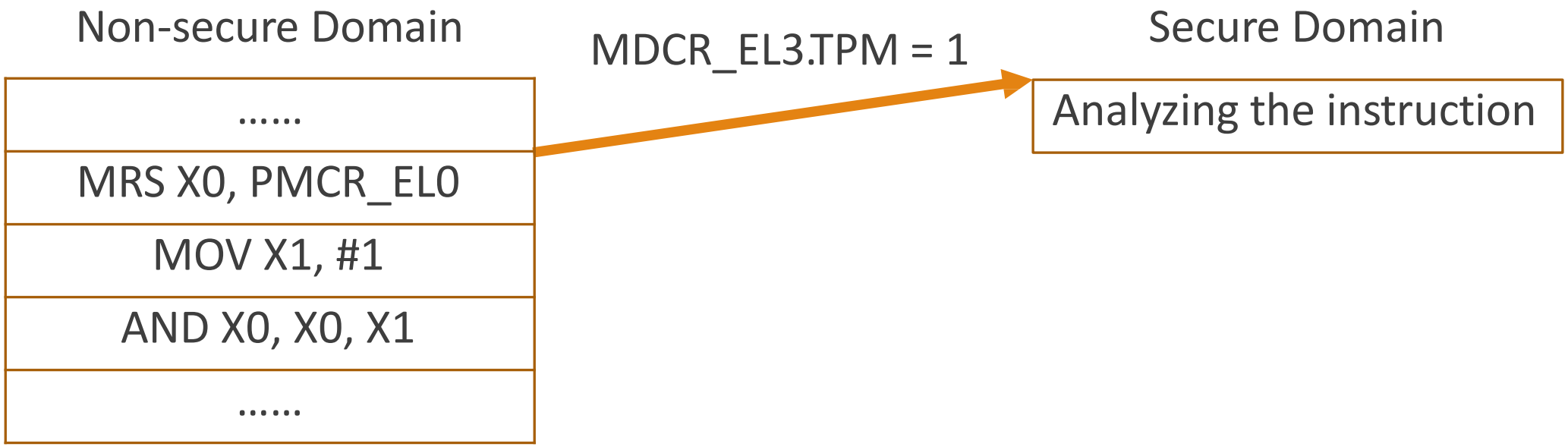
Hardware Traps

Non-secure Domain

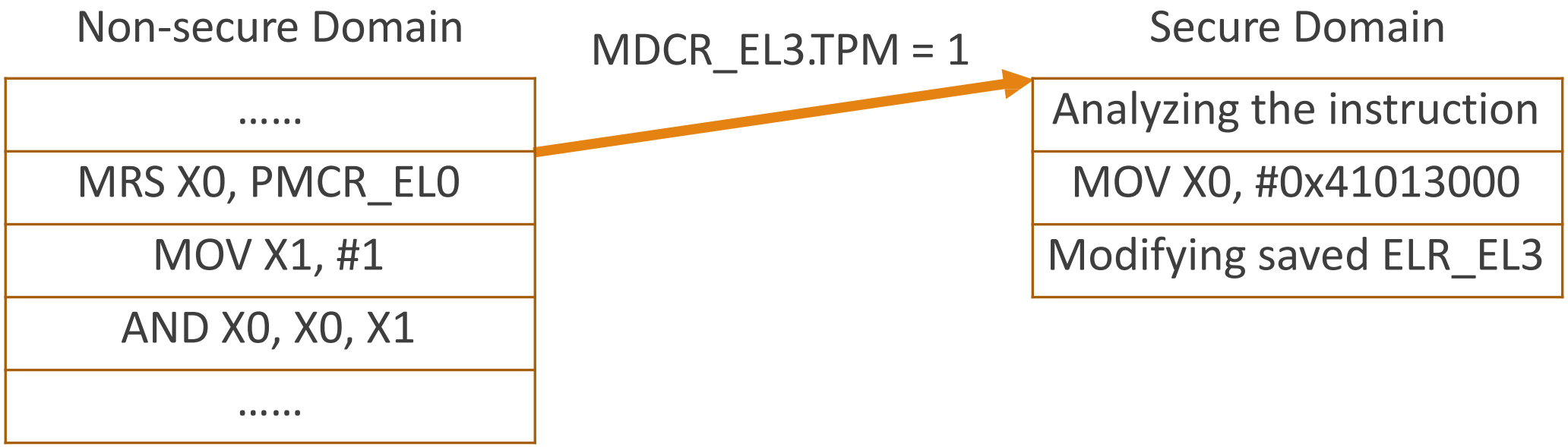
……
MRS X0, PMCR_EL0
MOV X1, #1
AND X0, X0, X1
……

The system-instruction interface makes the system registers readable via MRS instruction and writable via MSR instruction.

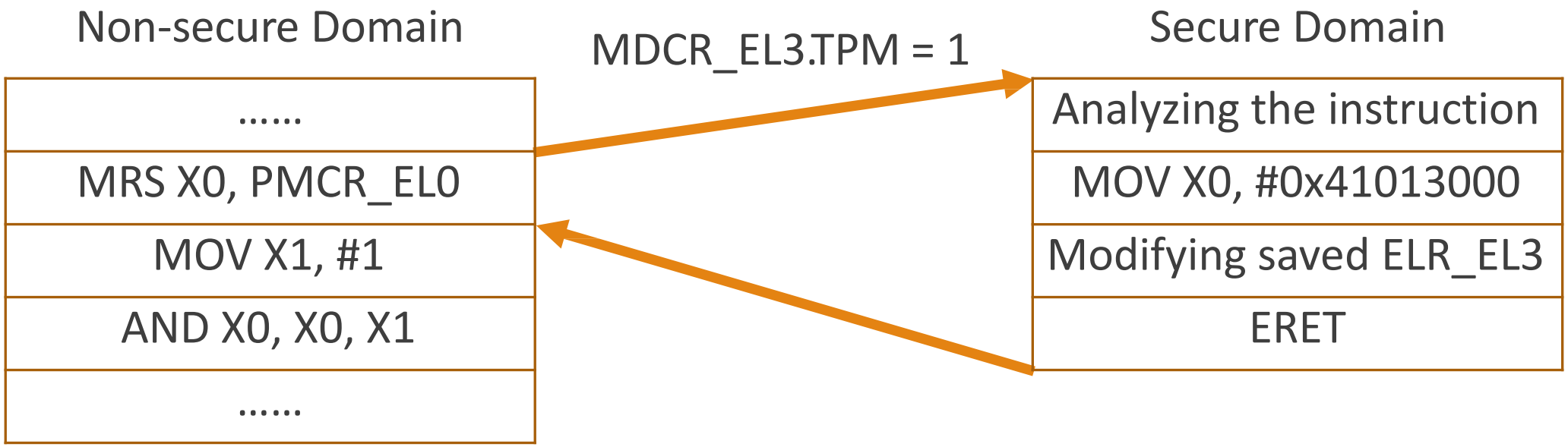
Hardware Traps



Hardware Traps



Hardware Traps



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?

Hardware-based approach provides better transparency.
To build a fully transparent system, 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 π
  - GNU Multiple Precision Arithmetic Library

	Mean	STD	#Slowdown
Base: Tracing Disabled	2.133 s	0.69 ms	
Instruction Tracing	2.135 s	2.79 ms	1x
System call Tracing	2.134 s	5.13 ms	1x
Android API Tracing	149.372 s	1287.88 ms	70x

Evaluation – Performance of the TS

- Performance scores evaluated by CF-Bench

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

Evaluation – Domain Switching Time

- Time consumption of domain switching (in µs)
  - 34x-1674x faster than MalT (11.72 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]

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.
  - NINJA can be easily transplanted to existing mobile platforms.