SATIN: A Secure and Trustworthy Asynchronous Introspection on Multi-Core ARM Processors

Shengye Wan, Jianhua Sun, Kun Sun, Ning Zhang, and Qi Li

William & Mary　George Mason University　Washington University in St. Louis　Tsinghua University

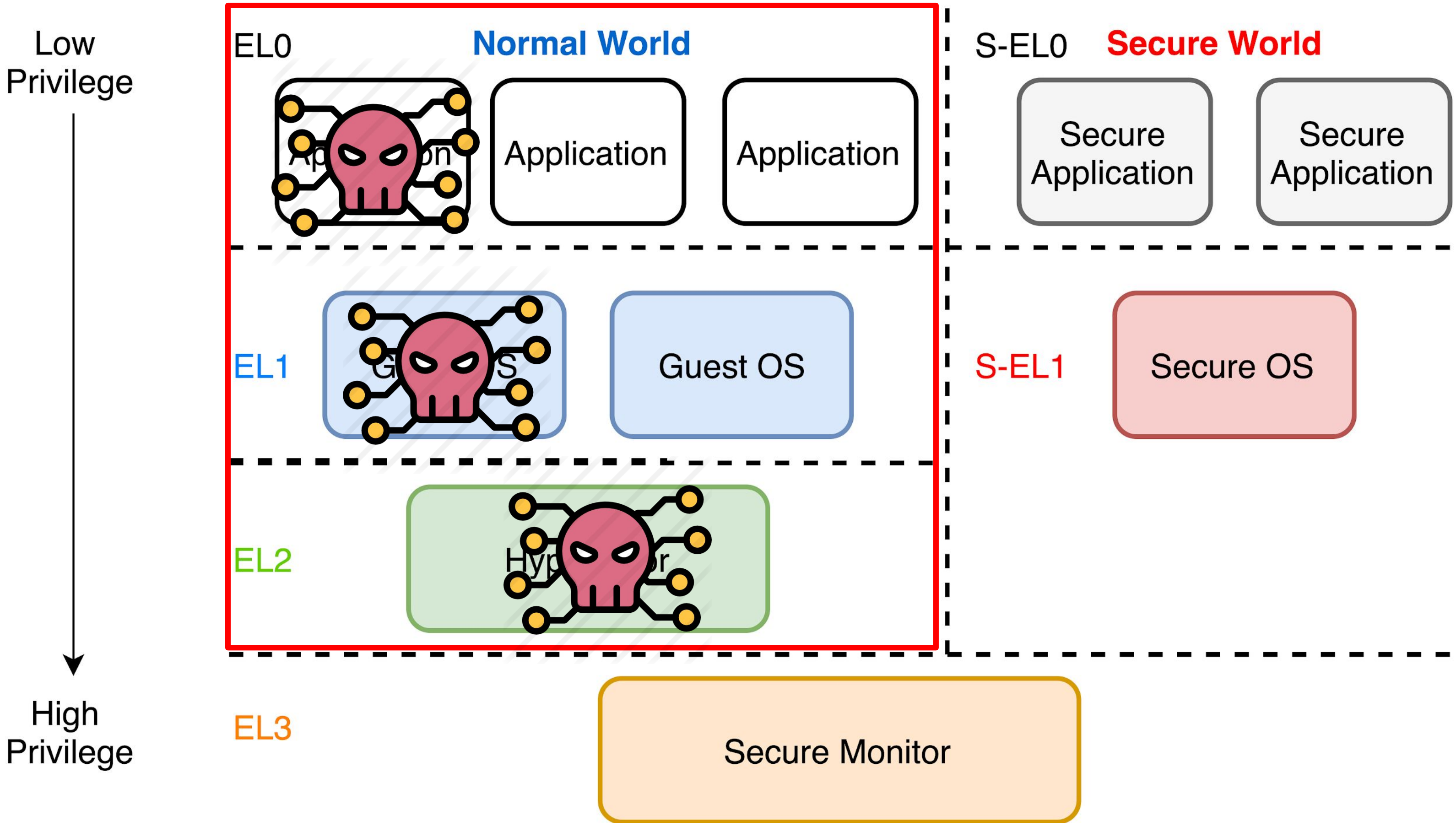
June/26/2019

Outline

- Background
  - TrustZone and Asynchronous Introspection
- New Evasion Attack on Multi-core Platform
  - Against TrustZone-based asynchronous introspection
- Defense
  - Secure TrustZone-based asynchronous introspection
- Takeaways

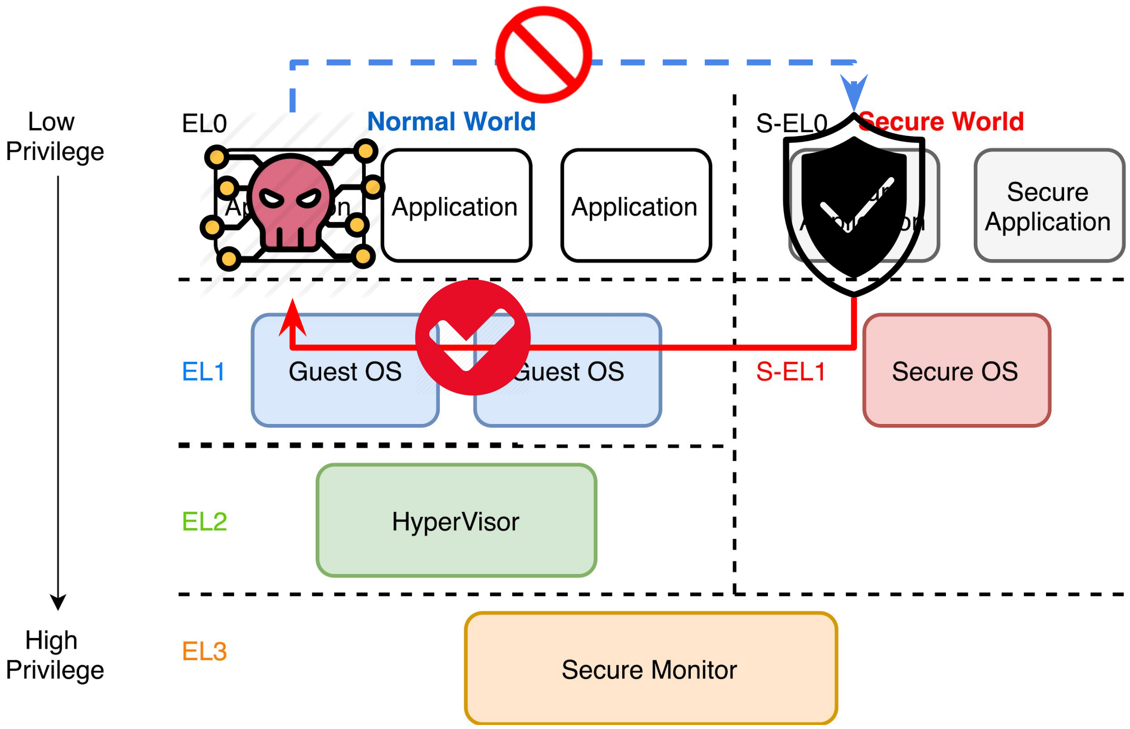
ARM TrustZone

Normal world is ***untrusted!***
Attackers may exist!



Inspect Normal World from Secure World

- TrustZone secure world has higher privilege
  - Accessing the system resources of the normal world such as memory, CPU registers, and peripherals

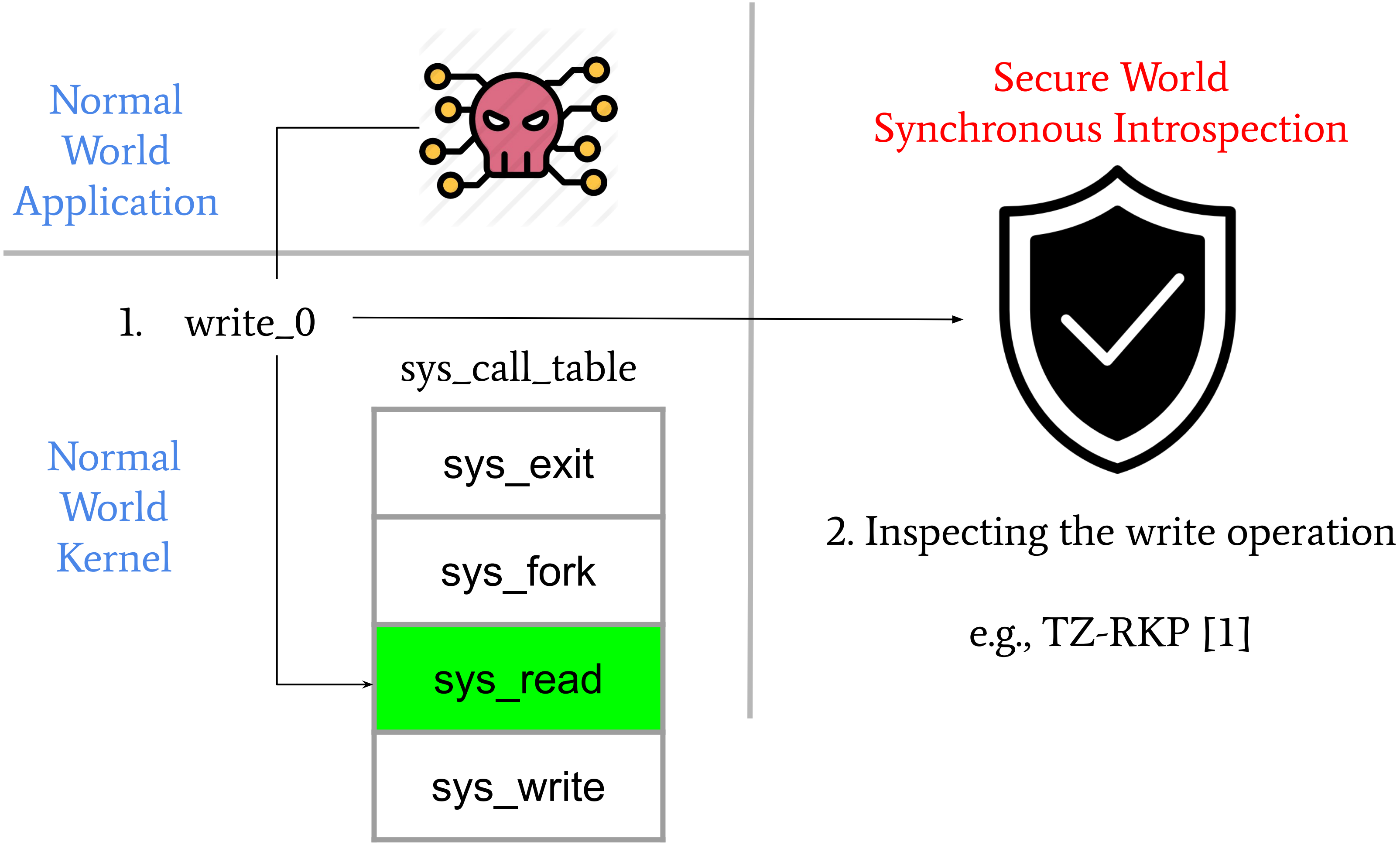


Introspection Techniques

1. Synchronous Introspection
    - Hooking the security-sensitive locations
    - Prevention

2. Asynchronous Introspection
    - Repeatedly analyzing the system snapshot
    - Detection

Trustzone-Based Synchronous Introspection



Normal World Application

1. write_0

sys_call_table

sys_exit
sys_fork
🟩 sys_read
sys_write

Normal World Kernel

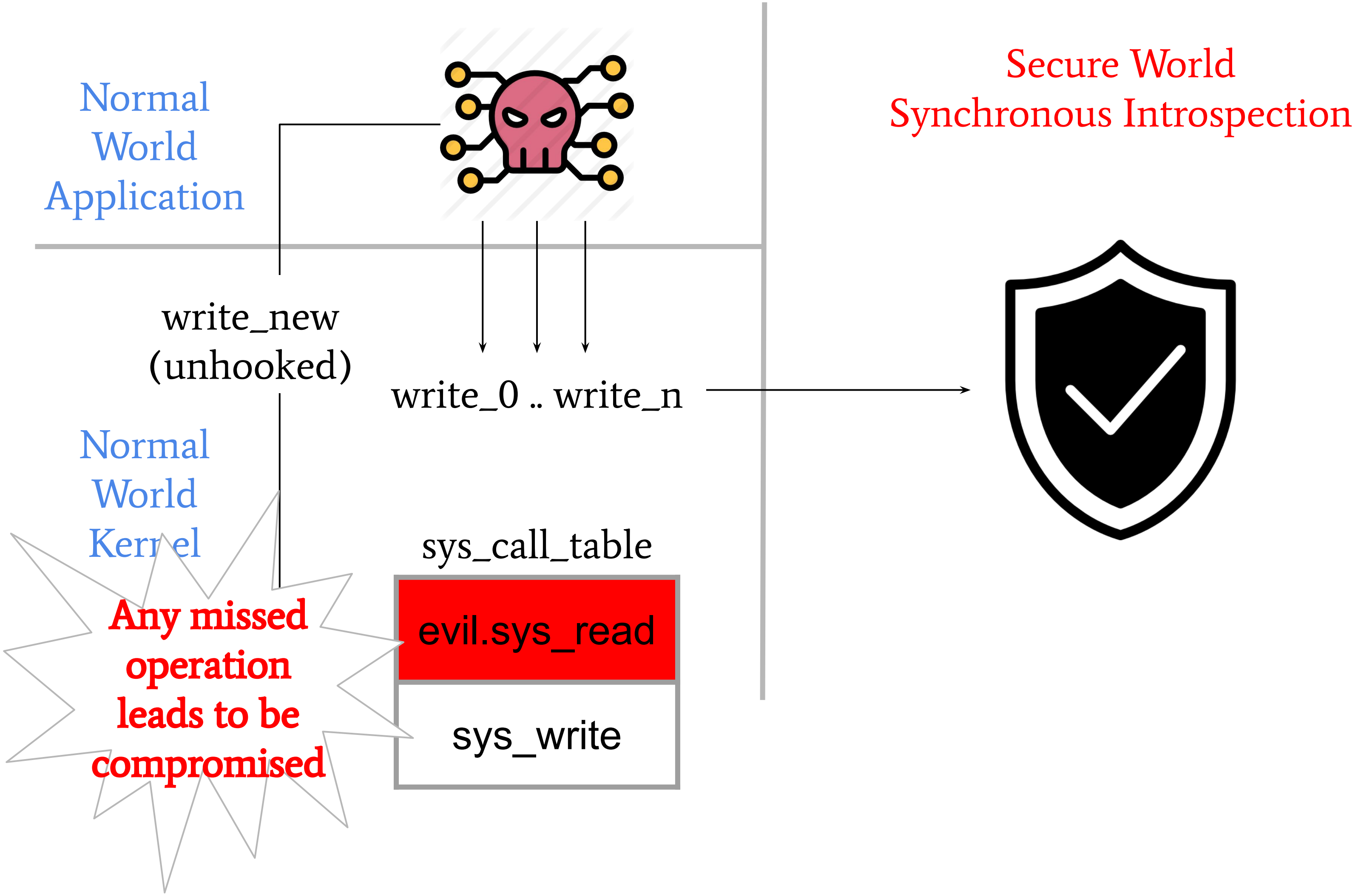
Secure World
Synchronous Introspection

2. Inspecting the write operation

e.g., TZ-RKP [1]

[1] Azab et al., “Hypervision across worlds: Real-time kernel protection from the arm trustzone secure world”

Synchronous Introspection Limitation



Normal World Application

write_new (unhooked)

write_0 .. write_n

Normal World Kernel

Any missed operation leads to be compromised

sys_call_table

evil.sys_read

sys_write

Secure World Synchronous Introspection

Synchronous Introspection Limitation

- Hard to hook up **all** security-sensitive locations
  - Cannot ensure the completeness of introspection
    - Unknown bugs
    - Bypass the checkpoints

- If the synchronous introspection is bypassed
  - Persistent stealthy attacks
  - E.g., Bypassing real-time kernel protection [2]

[2] Project Zero, “Lifting the (hyper) visor: Bypassing samsung’s real-time kernel protection”

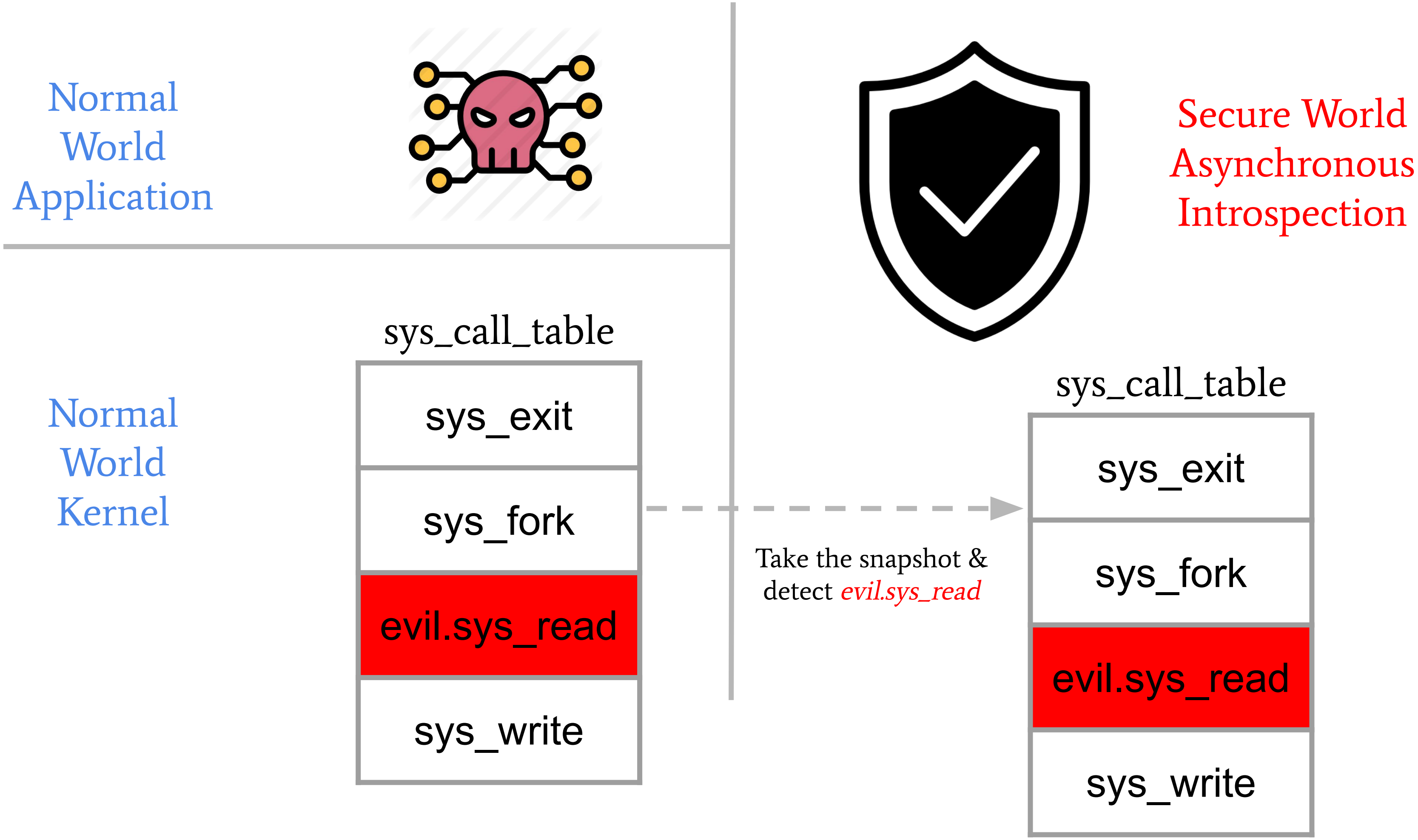
TrustZone-Based Asynchronous Introspection

- Detecting persistent stealthy attacks
- Two steps
    1. Taking a snapshot of memory along with CPU state information
    2. Analyzing snapshot to detect security policy violations
        - Checking the integrity of the invariant kernel code
        - Fine-grained security checking on dynamic kernel data structures

- Example: Samsung KNOX PKM (Periodic Kernel Measurement) [3]

[3] Samsung Electronics Co. Ltd., “White paper: An overview of the samsung knox platform”

TrustZone-Based Asynchronous Introspection



Normal World Application

Normal World Kernel

Secure World Asynchronous Introspection

sys_call_table

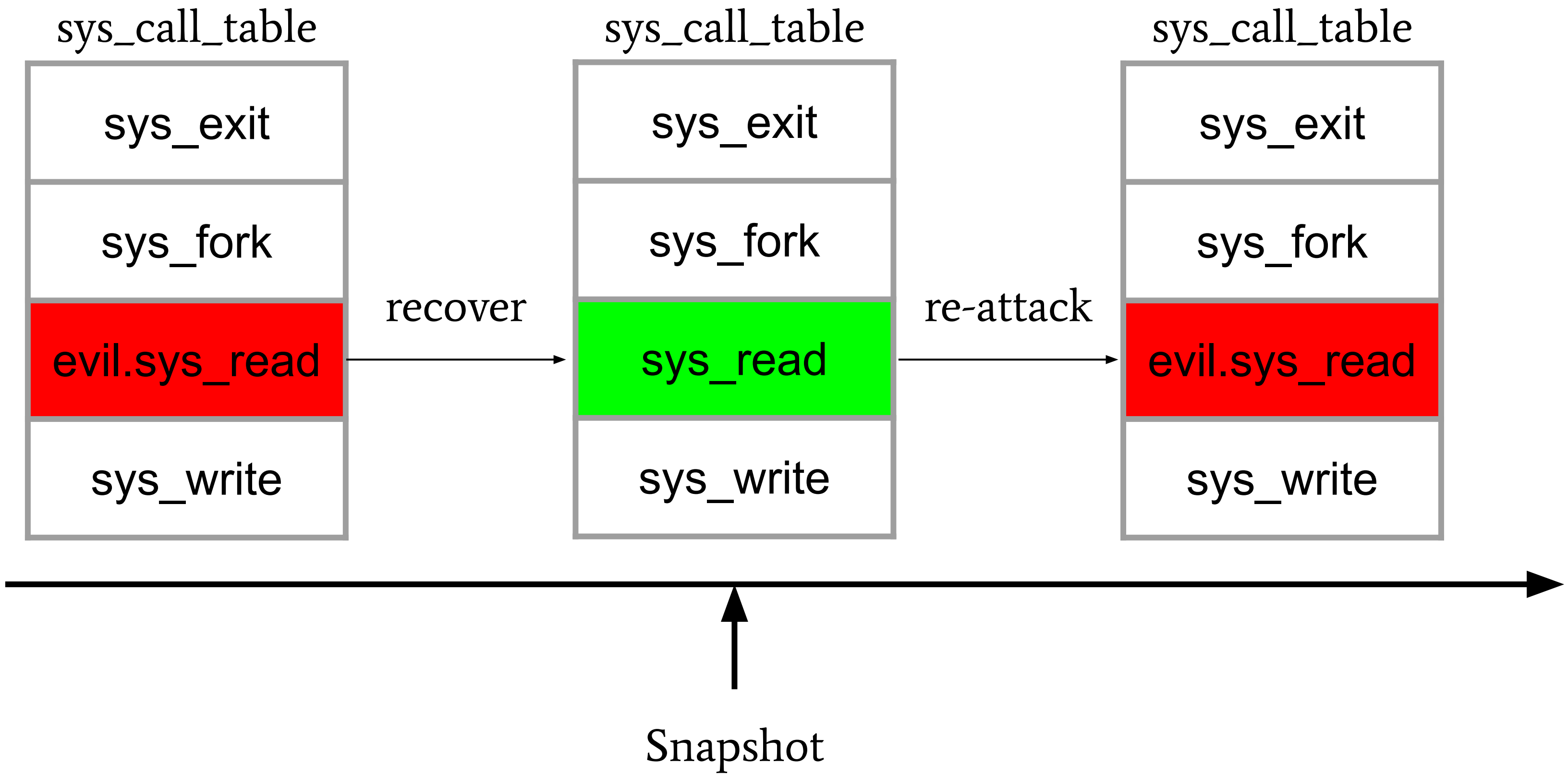
- sys_exit
- sys_fork
- evil.sys_read
- sys_write

Take the snapshot & detect *evil.sys_read*

sys_call_table

- sys_exit
- sys_fork
- evil.sys_read
- sys_write

Asynchronous Introspection Suffers Evasion Attack



Previous TEE-Based Asynchronous Introspection

- Single core asynchronous introspection in SMM [4,5]
  - No predictable pattern
  - When TEE is taking a snapshot, normal world is totally frozen
    - One core can only serve either TEE or normal world
    - Freezing is acceptable on single-core platform
  - Does not work on multi-core platforms

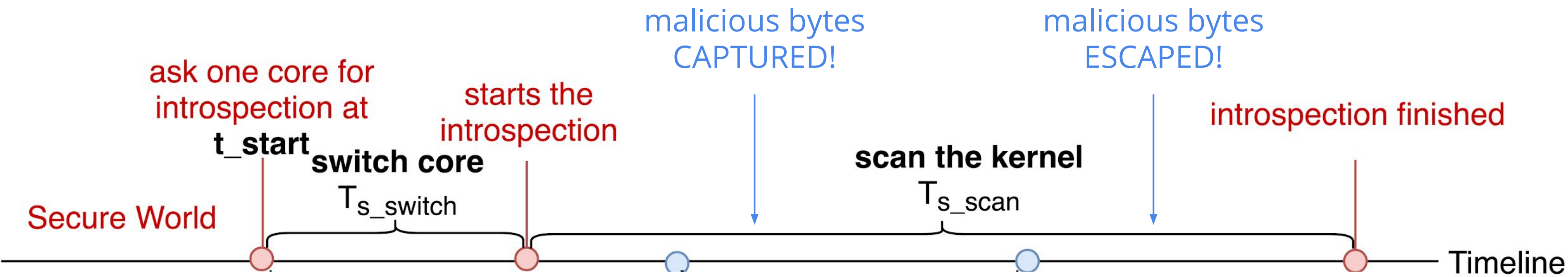
[4] Zhang et al., “Spectre: A dependable introspection framework via system management mode”
[5] Zhang et al., “Hypercheck: A hardware-assisted integrity monitor”

Challenges on Multi-Core Platform

- It is not practical to fully freeze the entire device
  - Needs to run both worlds' tasks simultaneously

- A new race condition is introduced
  - The attacker in normal world is active during introspection

Multi-Core Race Condition



Introspection covers entire kernel, while malicious byte can be anywhere

$$(T_{s\_switch} + T_{s\_scan}) \text{ v.s. } (T_{ns\_delay} + T_{ns\_recover})$$

Attacking Conditions

1. Probing when does the introspection start
   - Secure world resources are invisible to the normal world

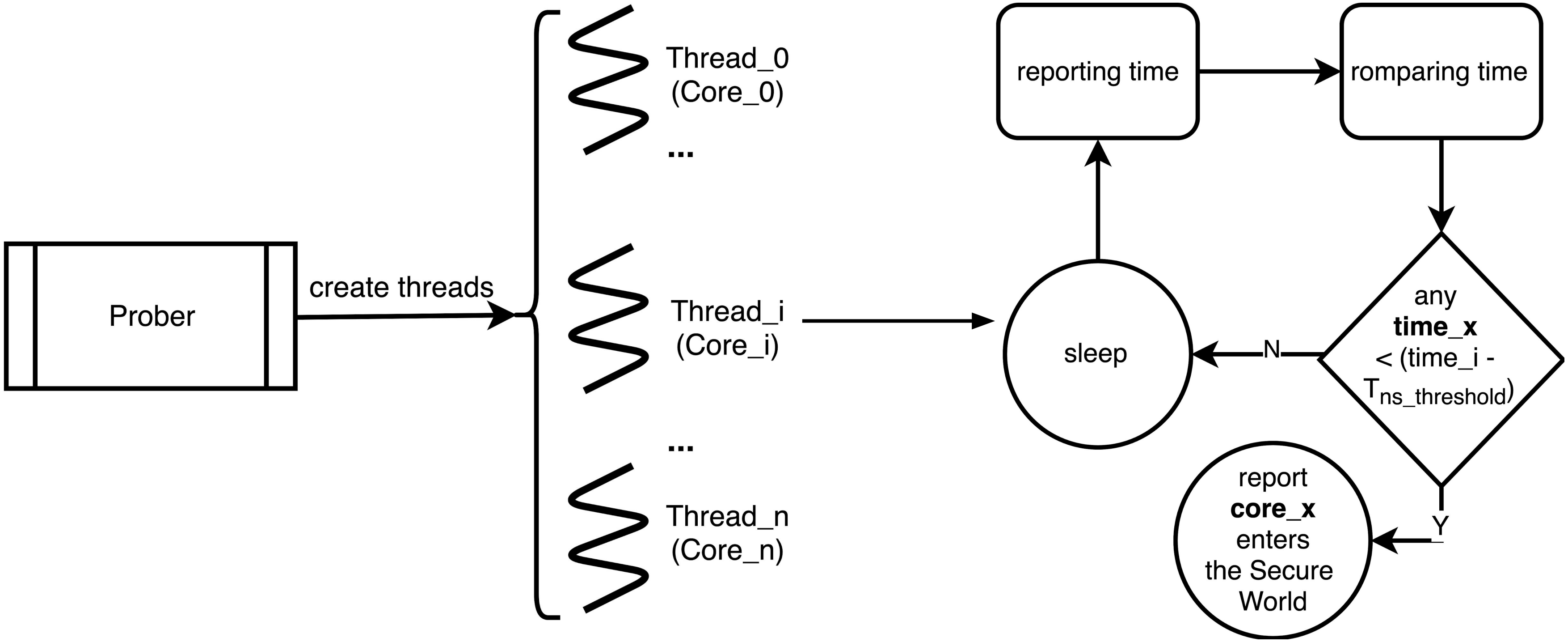
2. Evading fast
   - The malicious trace has to be cleaned before the snapshot being taken by secure world

TZ-Evader Design

- Two components
  1. Asynchronous introspection prober
     - Key idea: checking core availability
       - Secure world’s core is not available to normal world
       - Every core reports its availability on its own initiative
  2. A recoverable attack
     - Hide & re-apply

Asynchronous Introspection Prober

- Assigning one thread to each core
- Reporting the latest time
- Comparing one core’s latest time with other cores’



Asynchronous Introspection Prober Example

Asynchronous introspection is not working

core_id	reported time
core_0	1.3
core_1	1.2
core_2	1.3
core_3	1.4

- Threads cannot be controlled completely synchronized
- The time differences between any two threads have an upper limit
- We define the probing threshold T_{ns_delay} = 1.4 - 1.2 = 0.2

Asynchronous Introspection Prober Example

Asynchronous introspection is working on core_1

core_id	reported time
core_0	2.1
core_1	1.2
core_2	2.1
core_3	2.3

$$t_{core\_0} > (t_{core\_1} + T_{ns\_delay})$$

⇩

core_1 is hold by secure world

TZ-Evader Evaluation

- ARM Juno r1 development board
  - 4-core A53 processor, 2-core A57 processor
  - Rich OS: OpenEmbedded LAMP OS with kernel version lsk-4.4-armlt
  - Introspection checks the integrity of static kernel: 11,916,240 bytes
- Attacker
  - Hijacking the the GETTID system call (8 bytes)
  - Escaping from all secure world asynchronous introspections

TZ-Evader Observation 1

- $(T_{s\_switch} + T_{s\_scan})$ v.s. $(T_{ns\_delay} + T_{ns\_recover})$
- Probing one core's availability
  - $T_{ns\_delay} \approx 0.5ms$
- Probing five cores' availabilities
  - $T_{ns\_delay} \approx 2ms$

To maximum the attackers' time cost, secure world should not fix it CPU affinity

TZ-Evader Observation 2

- Worst cases for the attacker
    - T_{s_switch} : 3.6 us, T_{s_ccan} : 79.48 ms
    - T_{ns_delay} : 2 ms, $T_{ns\_recover}$: 6.13 ms
    - At the moment attacker recovers its trace:
        - Secure world only inspects 10% of the kernel

TZ-Evader is fast enough to recover attacks happen in **90% of the kernel** without being detected.

SATIN: A Secure and Trustworthy Asynchronous Introspection

- Self activation
    - Use the secure timer
        - Always invoke secure world to handle the interrupt
        - Do not engage normal world to invoke the introspection

- Random activation

SATIN: A Secure and Trustworthy Asynchronous Introspection

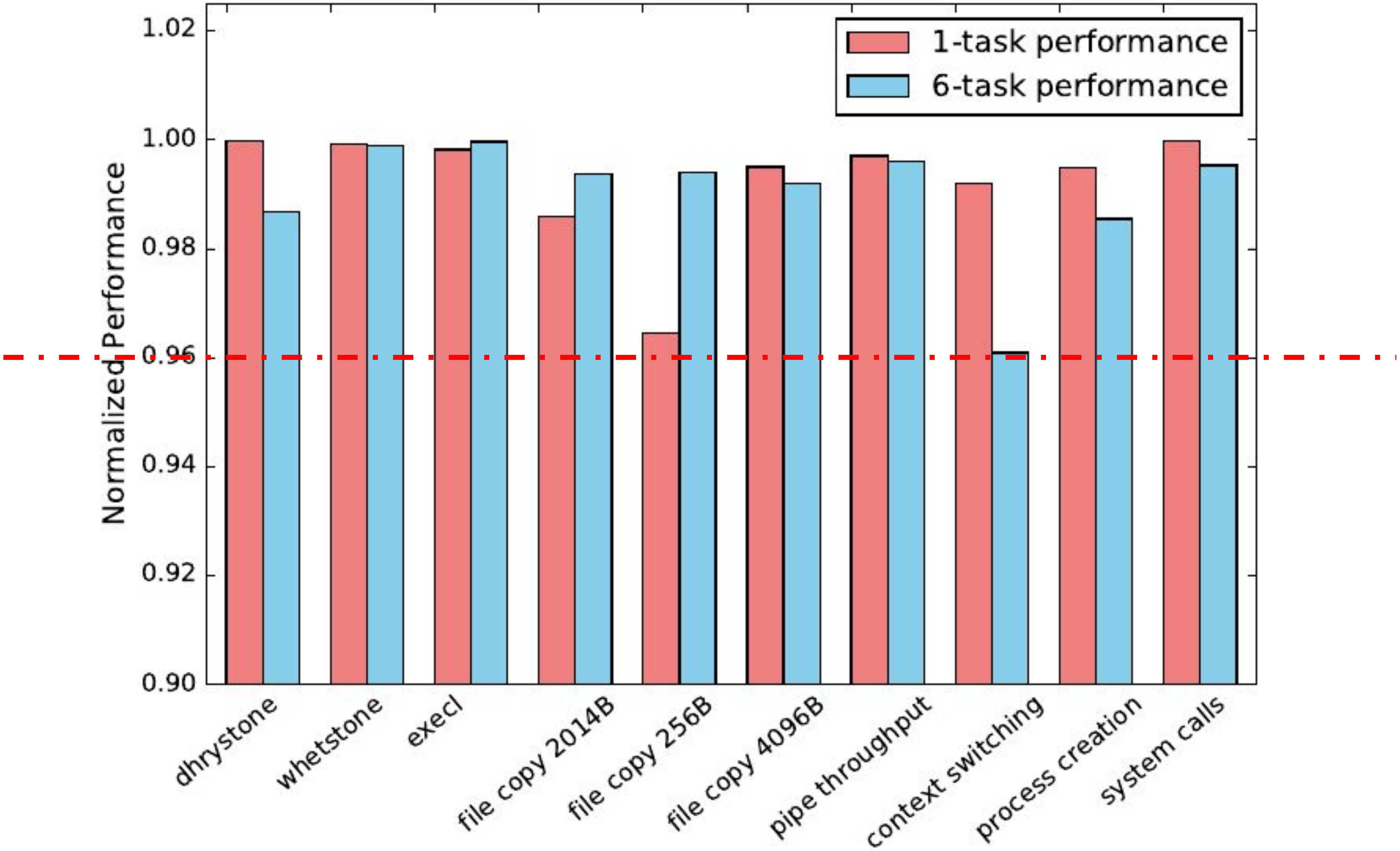
- Releasing the CPU core before normal world realizes it
  - Dividing the task into small sub areas
  - The time for inspecting each sub area should be shorter than
    - $T_{ns\_delay} + T_{ns\_recover} - T_{s\_switch}$

- Using all cores randomly
  - Increasing the difficulty of the normal world to conduct TZ-Evader

SAINT Performance

- Divide the normal world's kernel into 19 areas
  - Largest area: 876,616 bytes, smallest area: 431,360 bytes
- Inspecting entire kernel takes 152s in average
- TZ-Evader is 100% captured
- Performance downgradation (UnixBench)
  - 0.711% for single core task
  - 0.848% for 6 cores task

SAINT Overhead



Takeaways

1. We need TrustZone-based asynchronous introspection
2. It is challenging to inspect the normal world without freezing it
3. Core availability can expose the secure world running information
4. A secure introspection should mitigate all forms of evasion attacks

Thank you!

Q&A