Nighthawk: Transparent System Introspection from Ring -3

ESORICS 2019

Lei Zhou[1 2 *], Jidong Xiao[3], Kevin Leach[4], Westley Weimer[4], **Fengwei Zhang**[5 2 **], Guojun Wang[6]

[1] Central South University, China
[2] Wayne State University, USA
[3] Boise State University, USA
[4] University of Michigan, USA
[5] SUSTech, China
[6] Guangzhou University, China

* Work was done while visiting COMPASS lab at WSU; ** The corresponding author

Outline

- **Introduction and Background**
- Architecture of Nighthawk
- Design and Implementation
- Evaluation: Effectiveness and Performance
- Conclusion

Privilege Layers

Ring 3	User mode	virus
Ring 0	Kernel mode	rootkits
Ring -1	Hypervisor	rootkits
Ring -2	SMM	rootkits (SMM reload)

Defense Mechanism

How to defend against the attacks in each layer?

Defense Mechanism

How to defend against the attacks in each layer?

Deploy a defense at the a more privileged layer !

Existing Malware Detection

- **Virtualization based defensive approach (*ring -1*)**

Advantages ---- Full control of VM.

Limitations ---- High performance overhead and more likely to be a new target of attack.

Existing Malware Detection

- **Virtualization based defensive approach (*ring -1*)**

Advantages ---- Full control of VM.

Limitations ---- High performance overhead and more likely to be a new target of attack.

- **Hardware based defensive approach (*ring -2*)**

Advantages ---- Small TCB and lower layer.

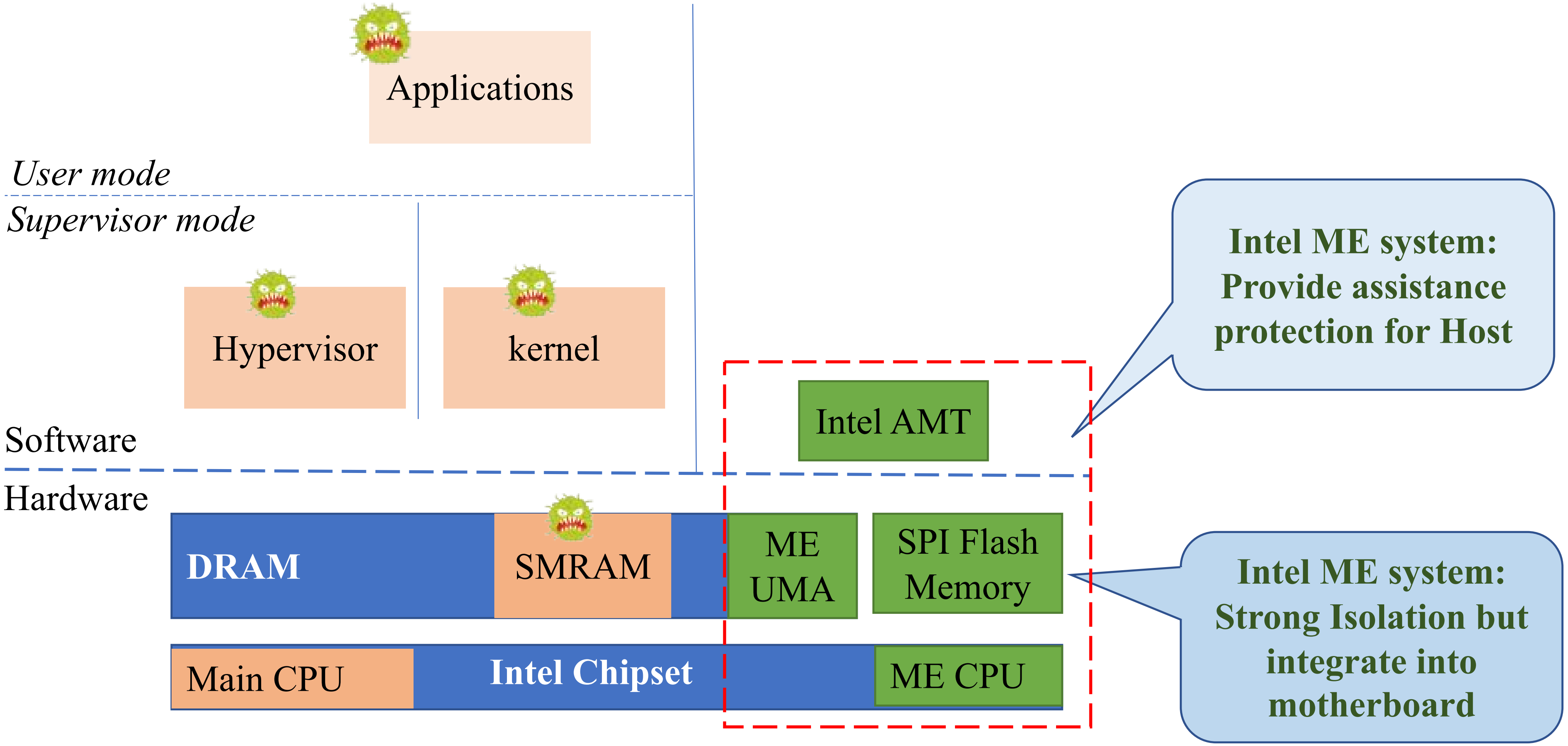
Limitations ---- Additional monitoring device or disturbing the normal system execution.

How to better defend against low-level attacks?

How to better defend against low-level attacks?

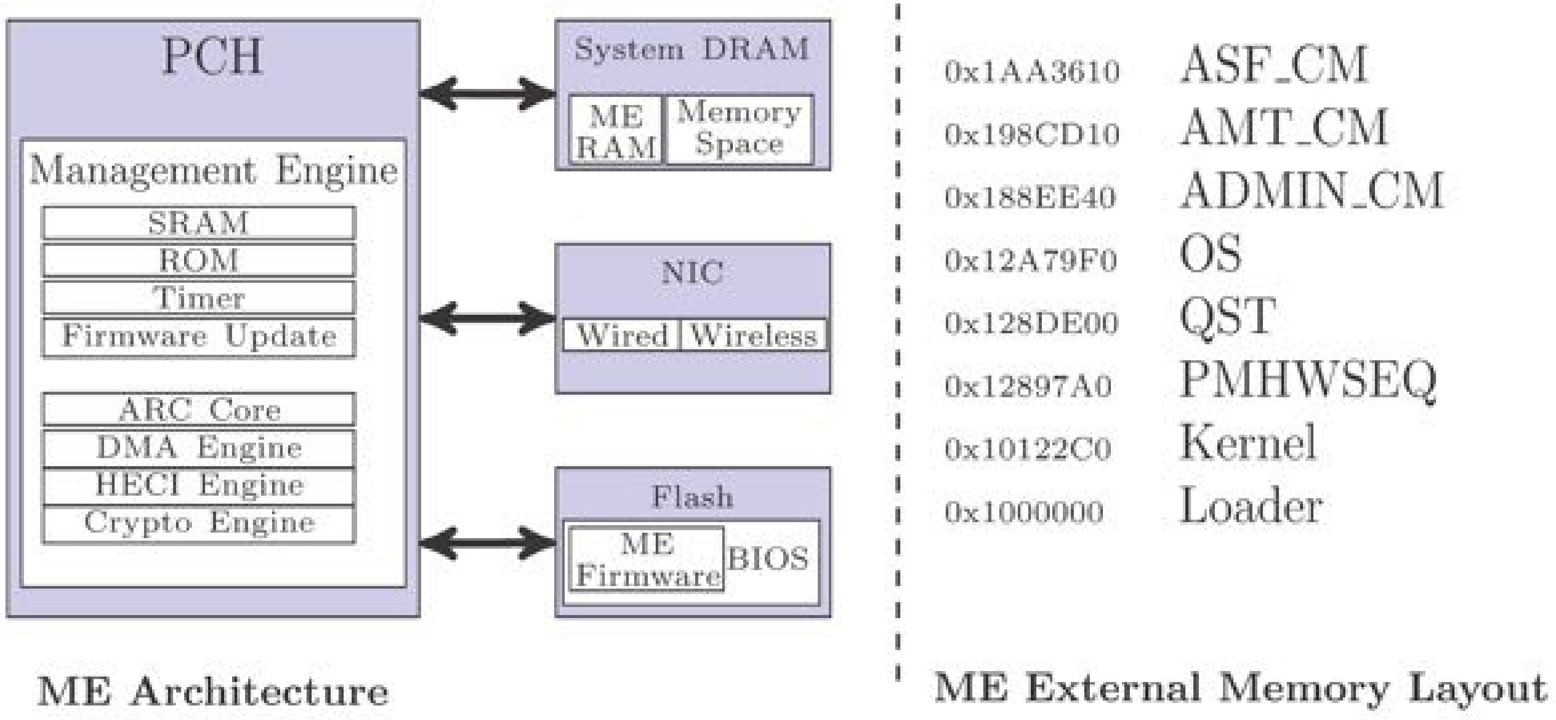
“Ring -3” ?

Higher Privilege System In Intel Architecture



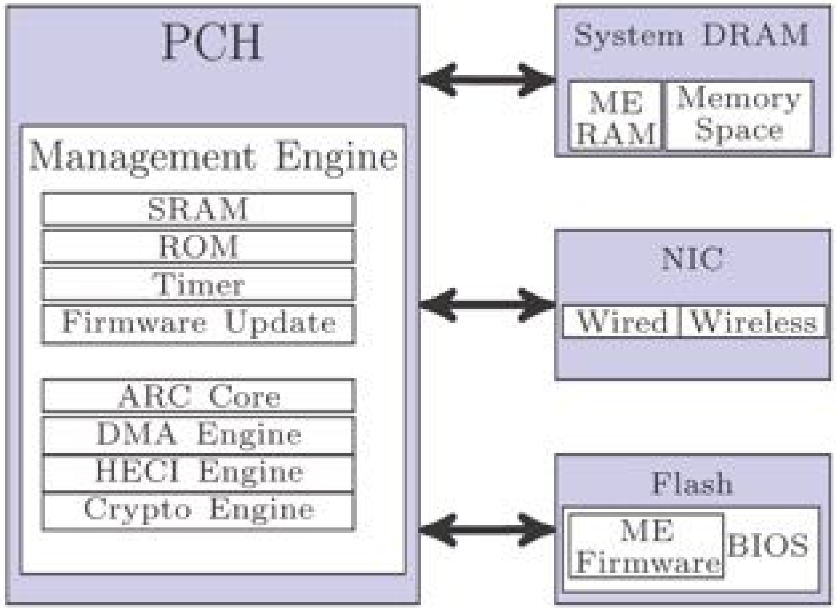
Understanding DMA Malware (DIMVA 2012)

Intel Management Engine



- ✓ ***No Extra Hardware Needed***
- ✓ ***Full Privilege***
- ✓ ***Small TCB***
- ✓ ***Transparency and low performance overhead***

Intel Management Engine



ME Architecture

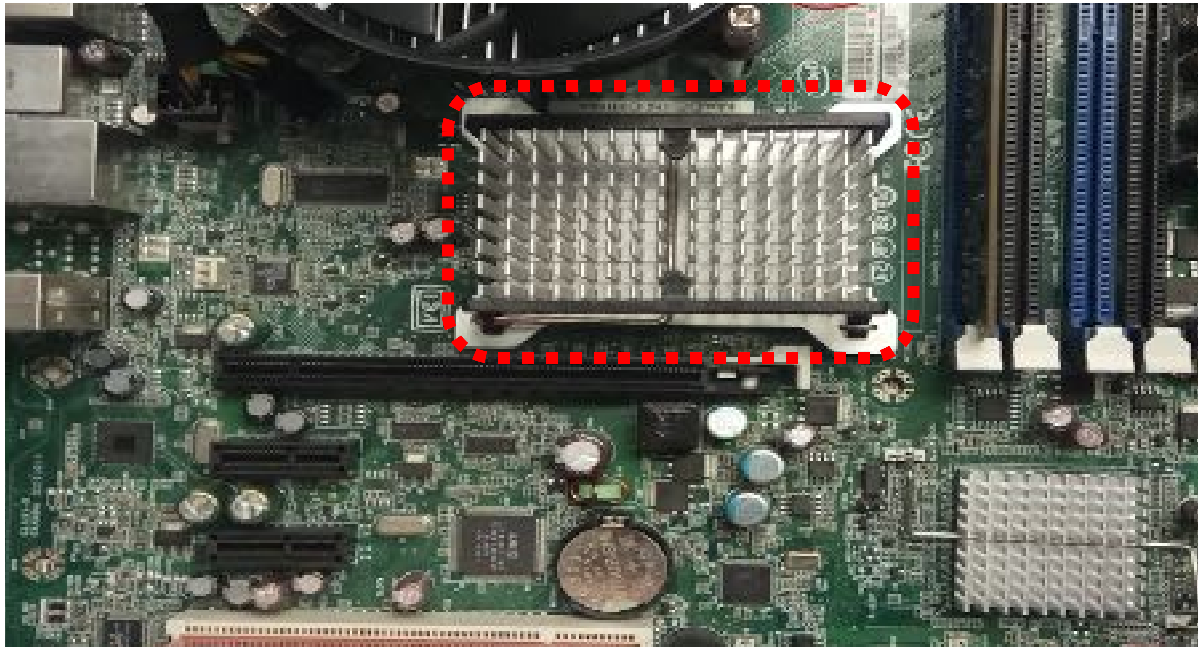
0x1AA3610	ASF_CM
0x198CD10	AMT_CM
0x188EE40	ADMIN_CM
0x12A79F0	OS
0x128DE00	QST
0x12897A0	PMHWSEQ
0x10122C0	Kernel
0x1000000	Loader

ME External Memory Layout

- ✓ ***No Extra Hardware Needed***
- ✓ ***Full Privilege***
- ✓ ***Small TCB***
- ✓ ***Transparency and low performance overhead***

However, IME related resources are not public to users

Location

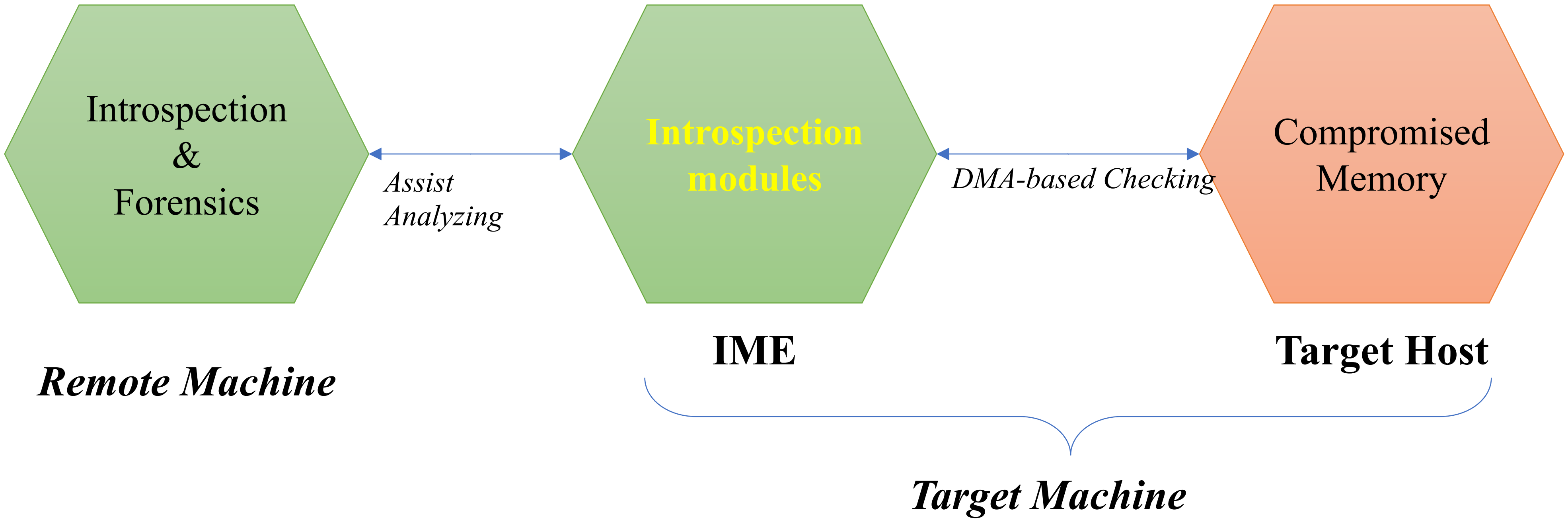


Microcontroller embedded in the PCH (older version in MCH)

Outline

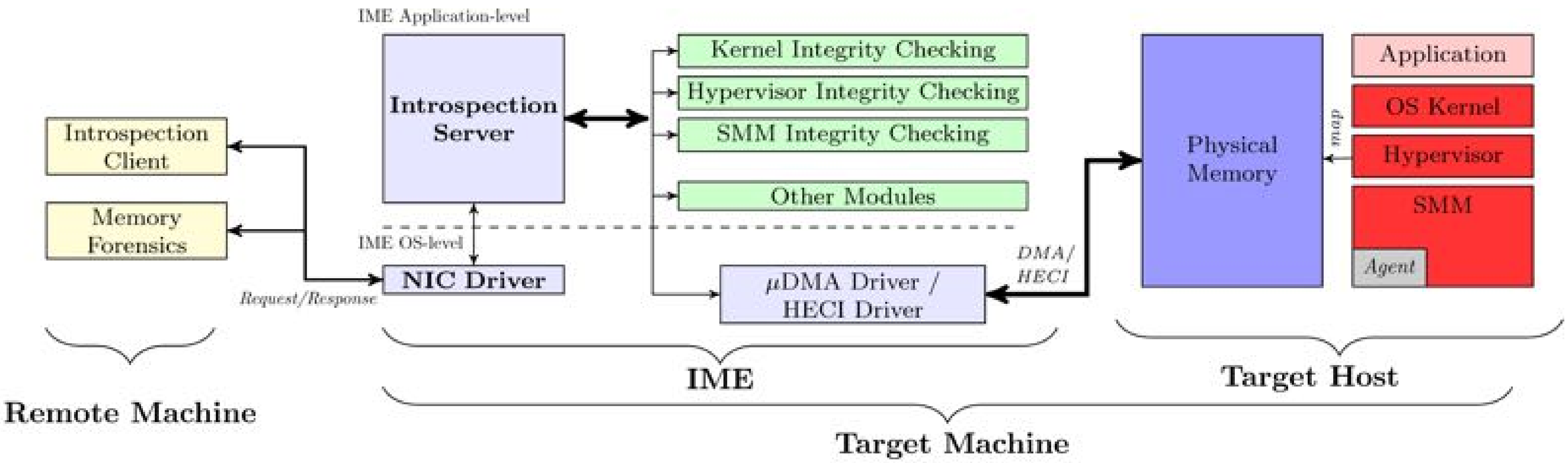
- Introduction and Background
- **Architecture of Nighthawk**
- Design and Implementation
- Evaluation: Effectiveness and Performance
- Conclusion

High-level Architecture of the Nighthawk



If we are able to add introspection code into IME system, we can check arbitrary host physical memory.

Details of Components in Nighthawk



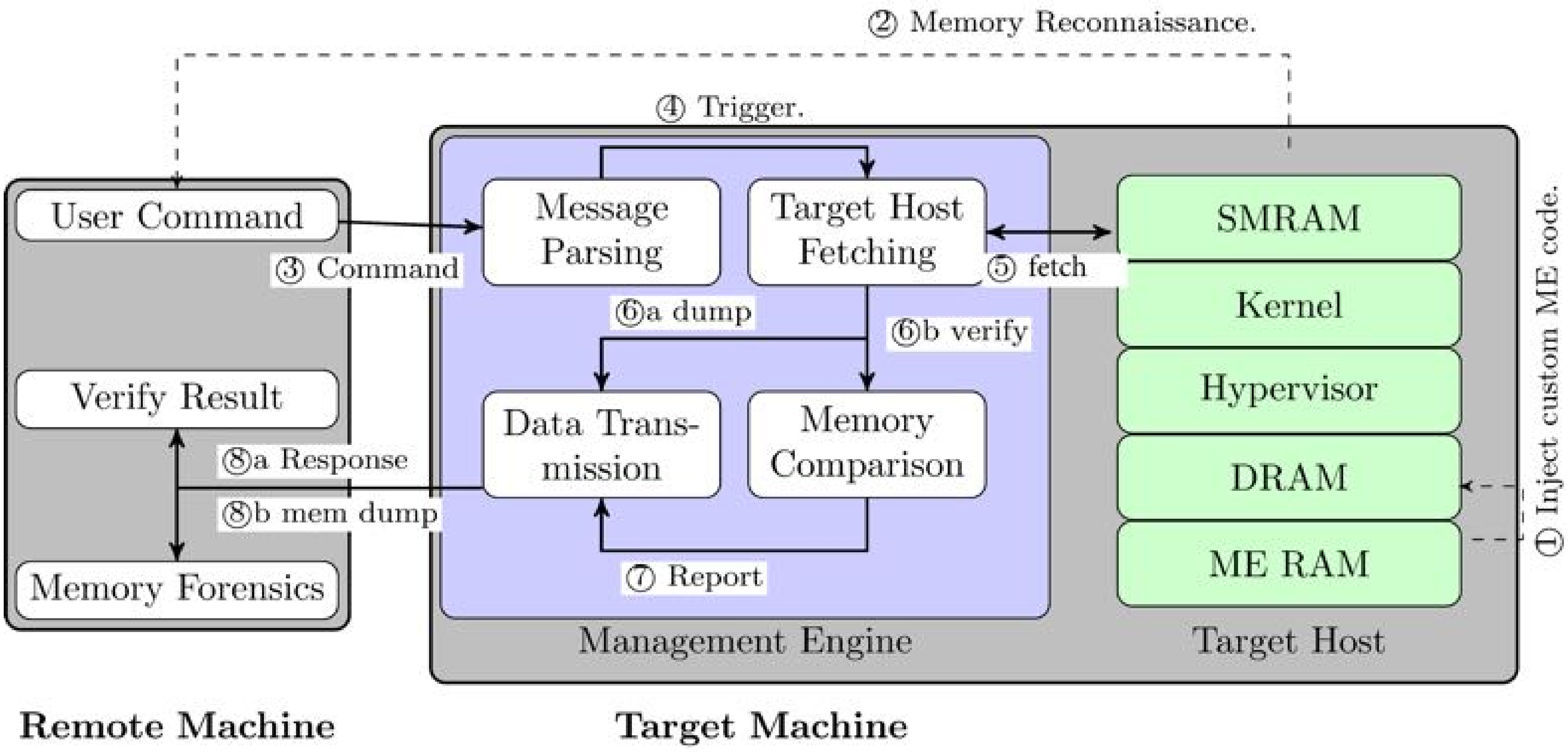
Outline

- Introduction and Background
- Architecture of Nighthawk
- **Design and Implementation**
- Evaluation: Effectiveness and Performance
- Conclusion

Nighthawk Design & Implementation

- Preparing the Target Machine
- Target Host Reconnaissance
- Measuring Integrity via Custom IME
- Command from Remote Machine

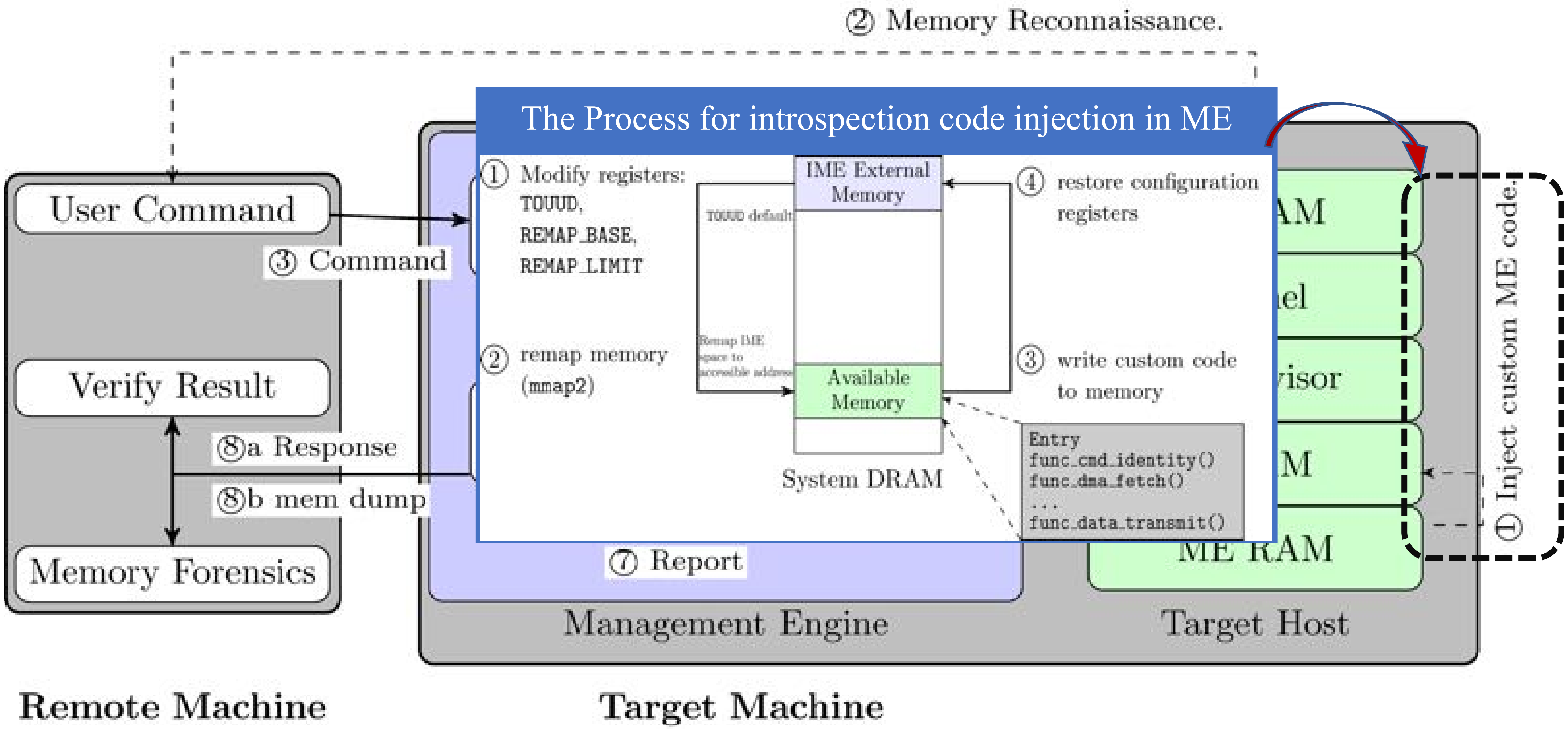
High-level Overview of the Implementation



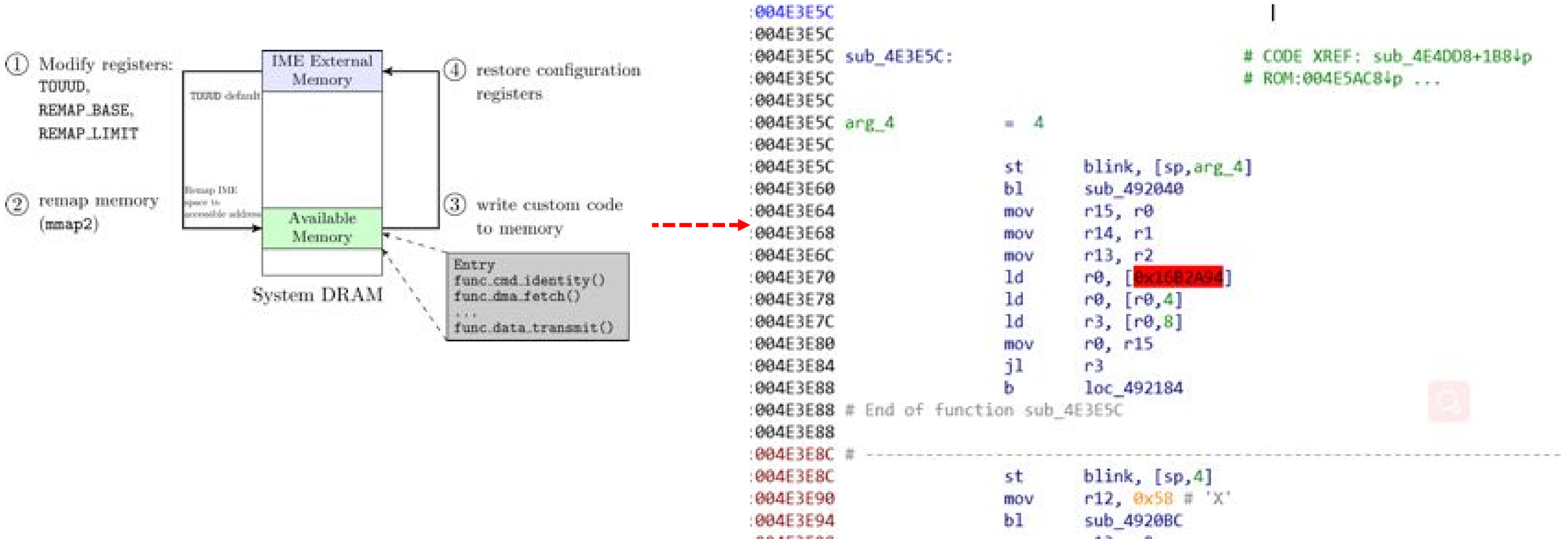
Nighthawk Design & Implementation

- *Preparing the Target Machine*
- Target Host Reconnaissance
- Measuring Integrity via Custom IME
- Command from Remote Machine

Preparing Target Machine (1) — Code Injection

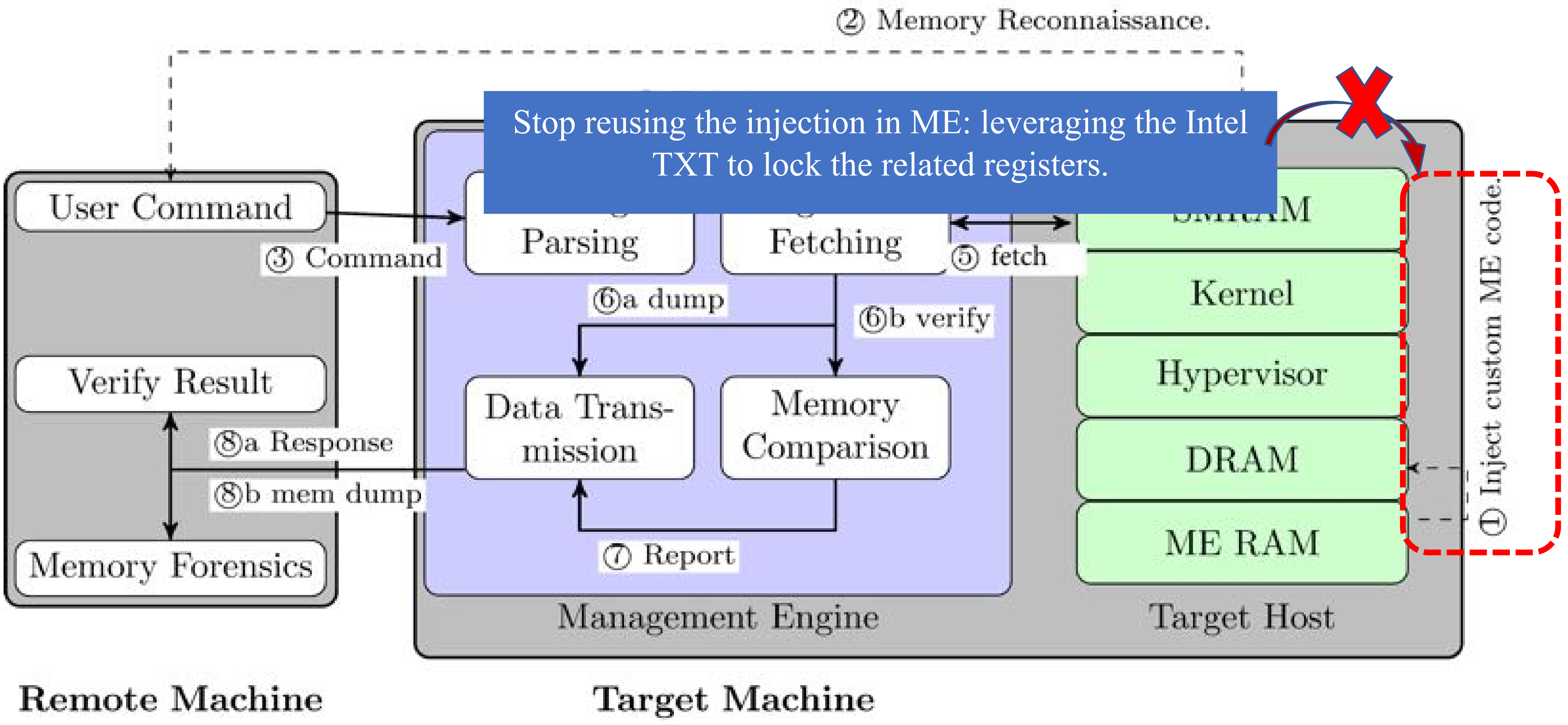


How to Inject the Introspection Code



Through Reverse engineering of the ME system code, we find the ideal function entry in which to inject the code.

Preparing Target Machine (2) — Stop Reusing Injection

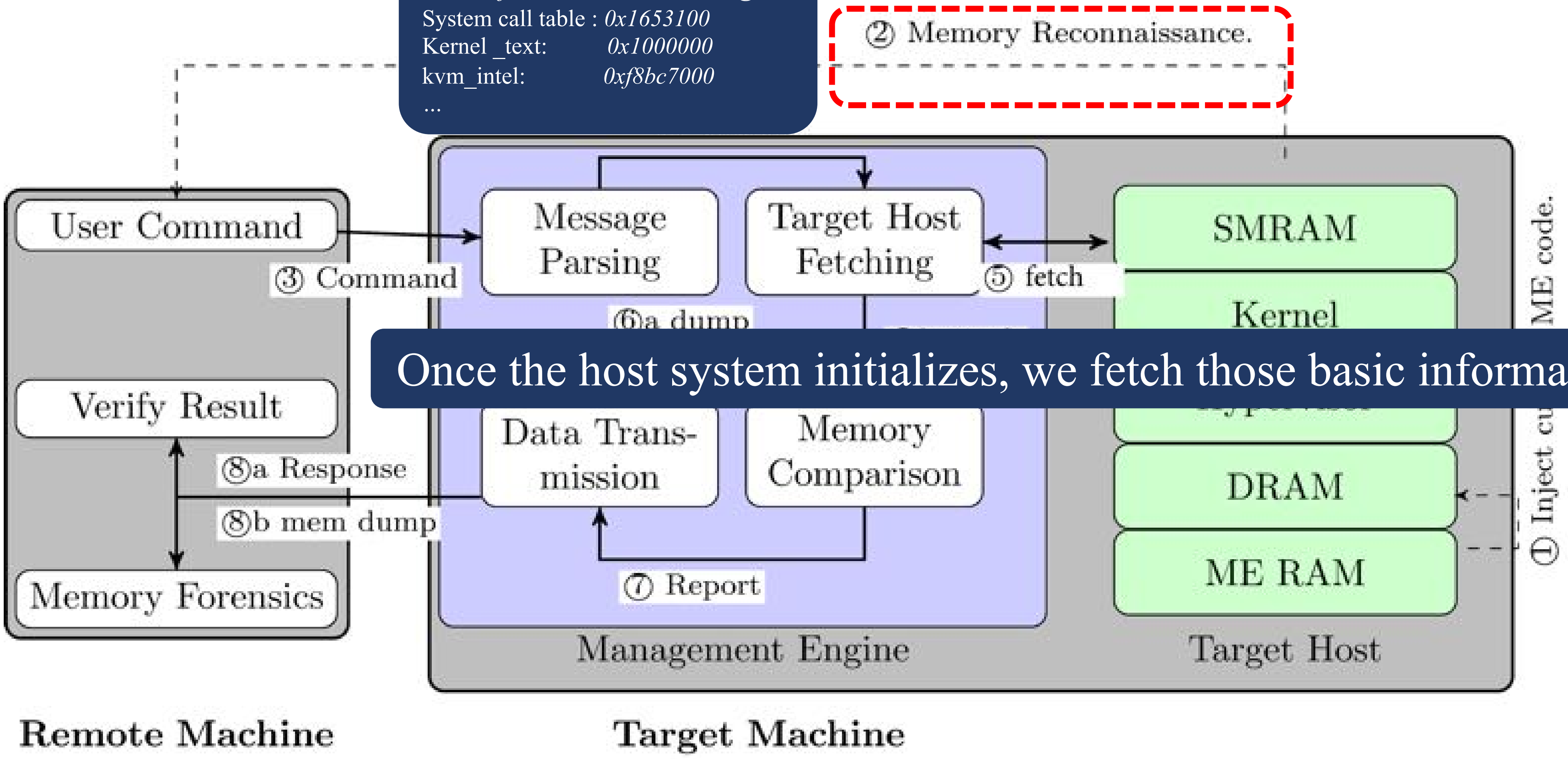


Nighthawk Design & Implementation

- Preparing the Target Machine
- ***Target Host Reconnaissance***
- Measuring Integrity via Custom IME
- Command from Remote Machine

Target Host Reconnaissance (1) — General Case

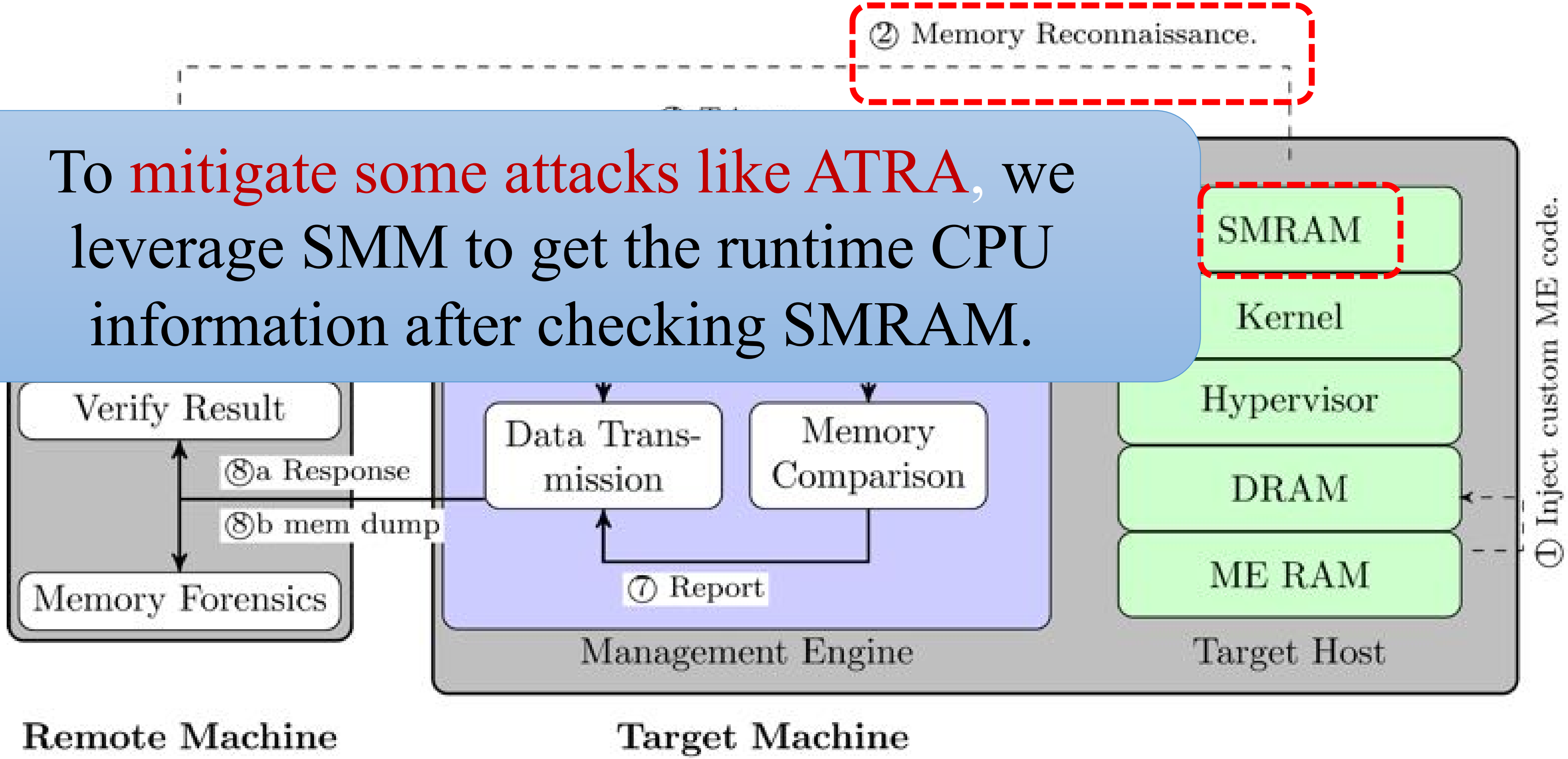
The information including:
System call table : *0x1653100*
Kernel _text: *0x1000000*
kvm_intel: *0xf8bc7000*
…



Once the host system initializes, we fetch those basic information.

Target Host Reconnaissance (2) — Special Case

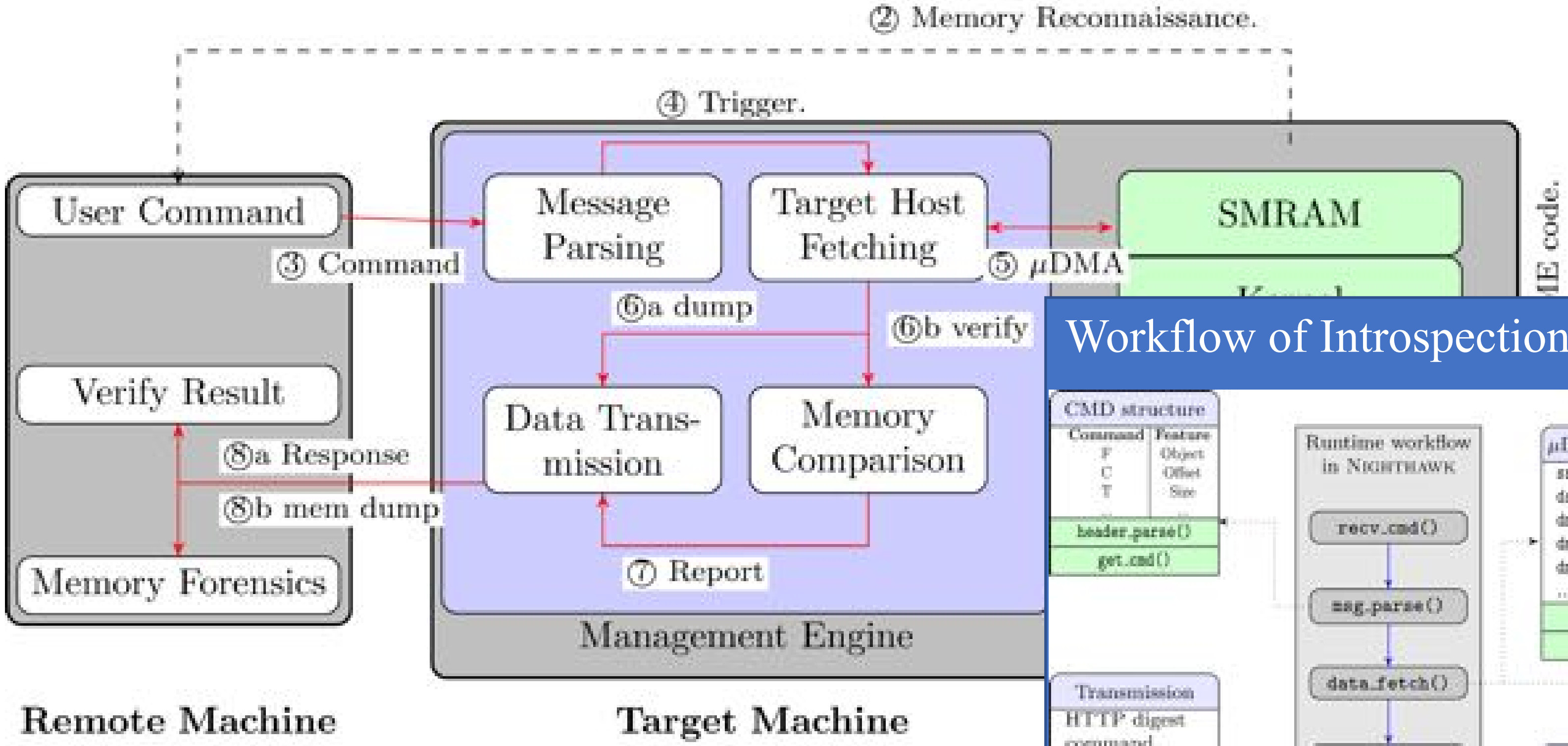
To mitigate some attacks like ATRA, we leverage SMM to get the runtime CPU information after checking SMRAM.

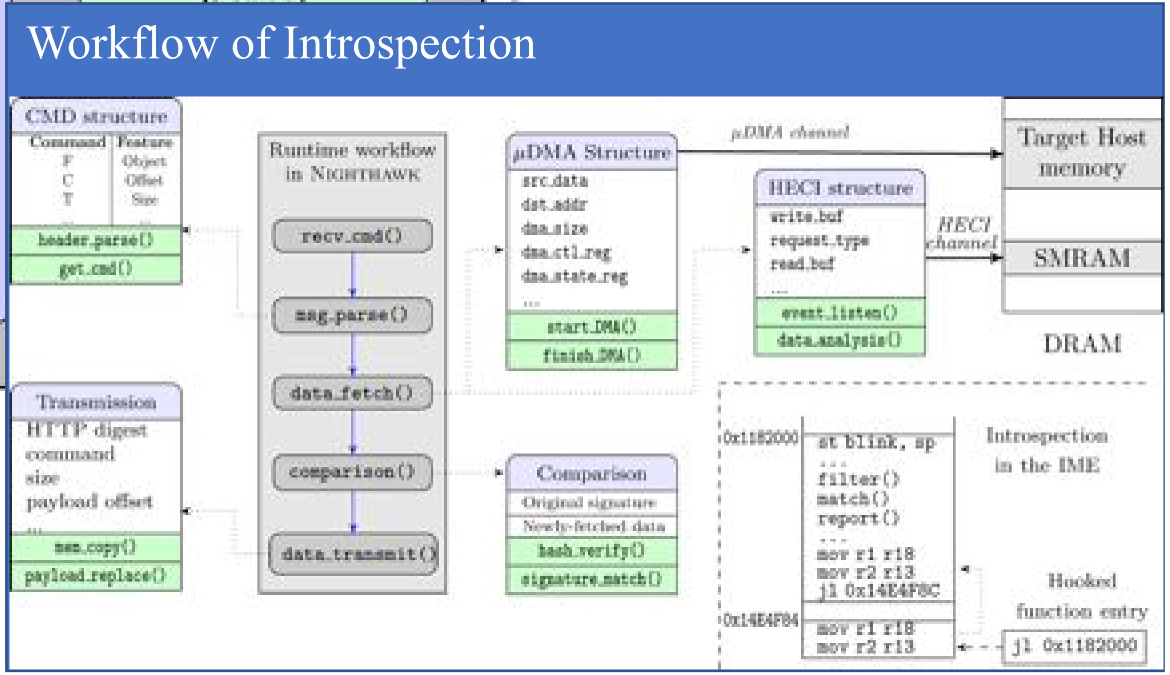


Nighthawk Design & Implementation

- Preparing the Target Machine
- Target Host Reconnaissance
- ***Measuring Integrity via Custom IME***
- Command from Remote Machine

Measuring Integrity via Custom IME

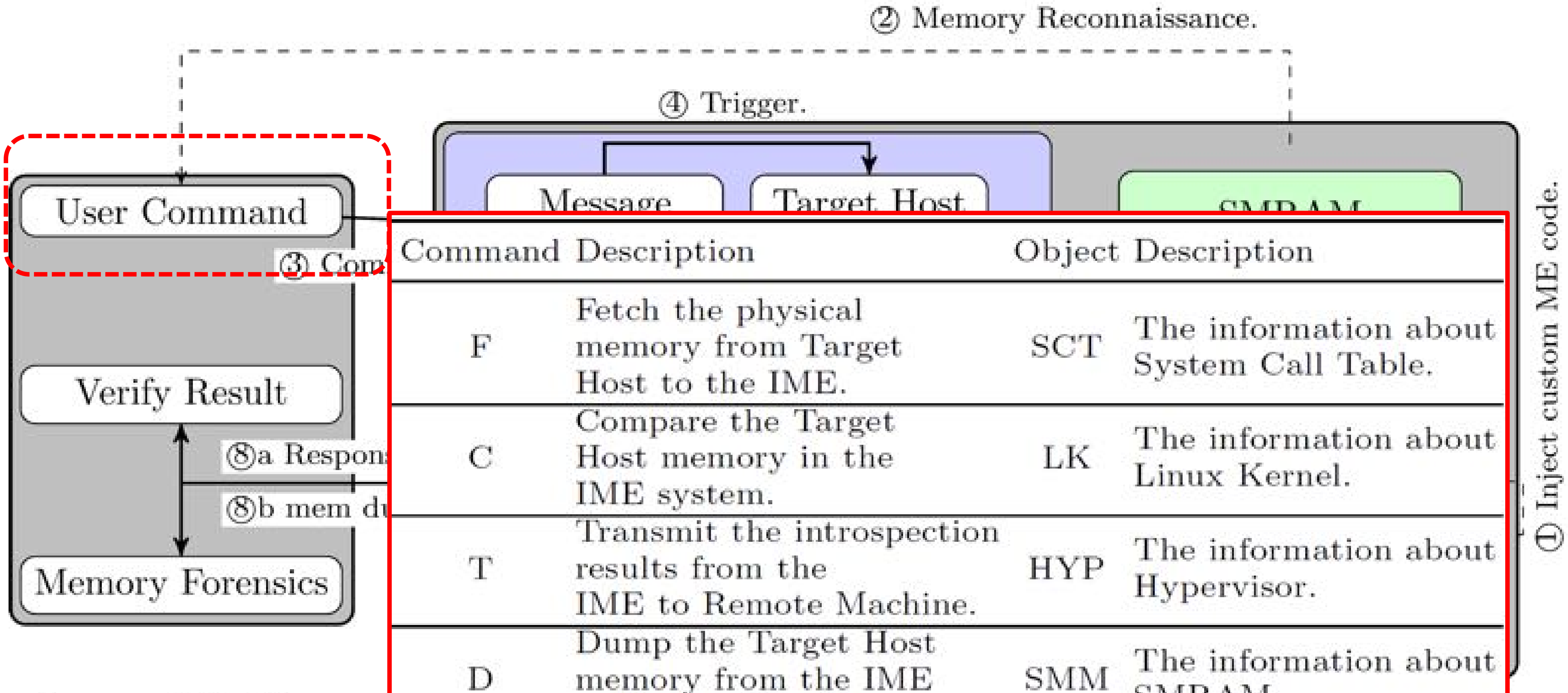




Nighthawk Design & Implementation

- Preparing the Target Machine
- Target Host Reconnaissance
- Measuring Integrity via Custom IME
- ***Command from Remote Machine***

Command from Remote Machine



Remote Machine

Command	Description	Object	Description
F	Fetch the physical memory from Target Host to the IME.	SCT	The information about System Call Table.
C	Compare the Target Host memory in the IME system.	LK	The information about Linux Kernel.
T	Transmit the introspection results from the IME to Remote Machine.	HYP	The information about Hypervisor.
D	Dump the Target Host memory from the IME to Remote Machine.	SMM	The information about SMRAM.

Outline

- Introduction and Background
- Architecture of Nighthawk
- Design and Implementation
- **Evaluation: Effectiveness and Performance**
- Conclusion

Evaluation

The test environment platform:

- ✓ Intel DQ35JO motherboard with 3.0GHz Intel E8400 CPU, ICH9D0 I/O Controller Hub and 2GB RAM.
- ✓ Intel e1000e Gigabyte network card for the network communication.
- ✓ We use an earlier BIOS version (JOQ3510J.86A.0933) for injecting code into ME.
- ✓ We run Ubuntu with the Linux kernel version 2.6.x to 4.x, along with KVM- and Xen-based Hypervisor.

Effectiveness--General Attacks

Target Object and Attacks

Object	Size (KB)	Time (s)
(General data)	1	0.258 ± 0.010
	4	0.261 ± 0.010
	64	0.267 ± 0.010
	256	0.387 ± 0.120
	2,048	3.06 ± 0.350
	3,096	4.67 ± 0.430
System Call Table	4	0.261 ± 0.010
Linux Kernel	6,466	9.75 ± 1.300
Hypervisor	336	1.31 ± 0.130
IDT	1	0.258 ± 0.010
Swapper_pg_dir	4	0.263 ± 0.010
SMRAM(unlocked)	128	0.383 ± 0.120
Random	10,240	15.4 ± 3.920

To simulate the attacking environment, we use existing rootkits for OS kernel, SMM, etc., installed in the target system.

We manually modify the memory content in kernel, Xen, KVM and SMM modules.

Through experiments, all attacks illustrated in this table have been detected by Nighthawk

Effectiveness -- Mitigating Special Attacks

ATRA Detection

We detect ATRA by testing for Page Global Directory and CR3 changes

Transient Attacks Detection

We simulate a transient attack using a toorkit-modified rootkit that changes the pointer address of the system call table.

Our results in the table show that Nighthawk can detect transient attacks in real world.

Execution Time (ms)	Attacks Detected Rate
< 8	<2.5%
12	7.5%
63	8.3%
123	22.5%
218	33.3%
437	68.3%
515	81.4%
643	92.1%
>700	100 %

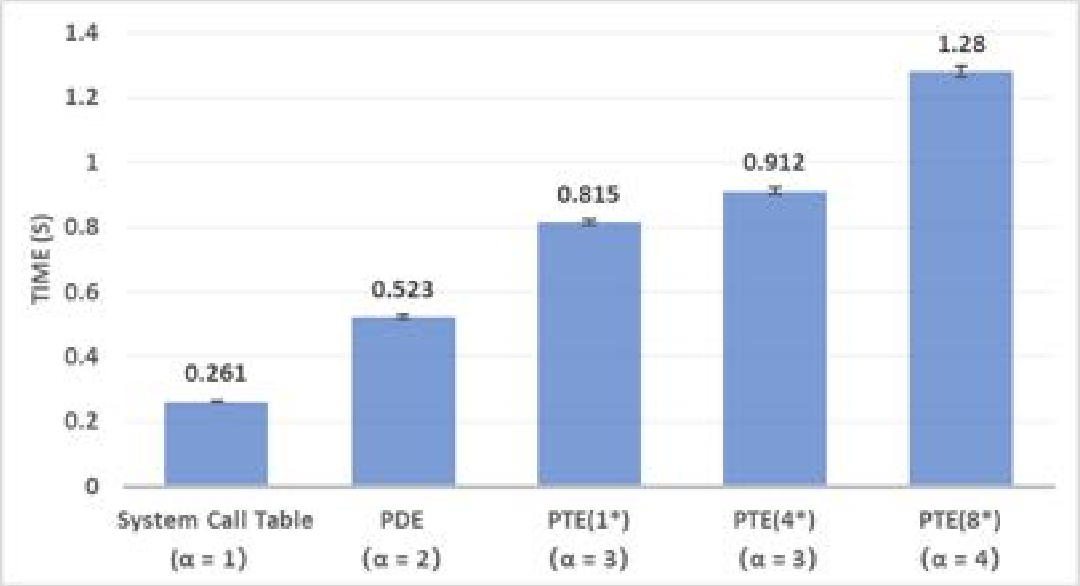
Performance Evaluation

DMA Fetching Overhead

Integrity Checking Overhead

Transmission Overhead

DMA Fetching Overhead



Time consumed by fetching data (Pages).
* represents the number of PTEs.
α represents accessing times.

Object	Size (KB)	Time (s)
	1	0.258 ± 0.010
	4	0.261 ± 0.010
	64	0.267 ± 0.010
(General data)	256	0.387 ± 0.120
	2,048	3.06 ± 0.350
	3,096	4.67 ± 0.430
System Call Table	4	0.261 ± 0.010
Linux Kernel	6,466	9.75 ± 1.300
Hypervisor	336	1.31 ± 0.130
IDT	1	0.258 ± 0.010
Swapper_pg_dir	4	0.263 ± 0.010
SMRAM(unlocked)	128	0.383 ± 0.120
Random	10,240	15.4 ± 3.920

Time consumed by DMA (User Cases).

Fetching data from host memory to ME memory

Memory Degradation Due To Introspection

With the benchmark test, the results show that Nighthawk has a very small performance impact to host.

Integrity Checking Overhead

- Time cost depends on the hash algorithm we choose.
  -- For 4KB memory page, it takes 7.3ms for checking under SDBM hash.

- Note that, for more complexity hash algorithm, e.g., sha1, it takes more time for checking.

- Compared to the fetching time, the checking time is very lower.

Comparison for Checking Overhead

With the SDBM hash verification test, we found the computing performance is much lower than it is in Host. For example, comparing a 6.3MB data, 25s is needed in ME, and 10 ms in Host.

Main factor: ME CPU core has a significantly lower computational capability.

We develop a CPU speed testing program, and the experimental result shows that the ME CPU executes approximately 15 million instructions each second (Meanwhile, billions per second on regular CPUs).

Transmission Overhead

- For a small message(<1*KB*), takes 228*ms* on average to pass the data.

- For a dumping data (i.e., > 64*KB*), we divide the data into multiple packets and transmit via multiple messages. e.g., 64KB data takes 4.9*s*.

Performance Evaluation Summary

Object	Size (KB)	Data Fetching Time (s)	Comparison Time (s)	Data Transmission Time (s)	Total Time (s)
System call table	4	0.26±0.010	0.007±0.001	0.224±0.030	0.50±0.030
kvm_intel.ko	336	1.31±0.130	0.601±0.010	0.231±0.030	2.14±0.150
PDE	4	0.52±0.010	0.007±0.001	0.230±0.030	0.76±0.040
SMRAM(unlocked)	128	0.39±0.150	0.320±0.005	0.228±0.030	0.94±0.200

Conclusion

- **Nighthawk—a transparent introspection framework**
  - Leveraging Intel ME
  - High privilege: ring -3
  - Small TCB
- **Attack scenarios**
  - Real-world attacks against OS kernels, type-I and type-II hypervisors, and unlocked system management RAM
- **Introducing almost zero overhead**

Thank you! Questions?

zhangfw@sustech.edu.cn

https://fengweiz.github.io/