When Hardware Meets Software: A Bulletproof Solution to Forensic Memory Acquisition

[1]Università degli Studi di Milano

[2]Royal Holloway, University of London

Alessandro Reina[1] Aristide Fattori[1] Fabio Pagani[1]
Lorenzo Cavallaro[2] Danilo Mauro Bruschi[1]

28^{th} Annual Computer Security Applications Conference

Digital Investigation

Full System Memory Dump

Acquisition of volatile memory is an essential procedure in digital forensic analysis and incident response.

Run-Time Information

- processes;
- network connections;
- open files;
- unencrypted data;
- passwords;
- malware;
- ...

Full System Memory Dump

Challenge

Tampering of the volatile memory during a dump invalidates the collected evidence.

Full System Memory Dump

Challenge

Tampering of the volatile memory during a dump invalidates the collected evidence.

Requirements

- *atomicity*: dump must represent the content of the memory at a single instant in time
- *reliability*: must be able to detect tampering or corruption of the dump
- *availability*: solutions must be OS and device independent

Limitations of Current Solutions: Hardware

- *PCI*: prior-installation requirement implies reduced usability
- *FireWire*: resolves the usability problem

Neither approach satisfies:

✗*atomicity*	the CPU is not *frozen*
✗*reliability*	bus scanning; subjects to DMA attack
✗*availability*	driver required

Limitations of Current Solutions: Software

- *virtual device* (e.g., `/dev/mem`):
  - ✗*atomicity* the CPU is not *frozen*
  - ✗*reliability* tool loaded in memory to be run
  - ✗*availability* OS dependent
- *hypervisor*:
  - ✓*atomicity* can freeze the *guest* and perform the dump
  - ✗*reliability* changes in memory due to hypervisor loading
  - ✗*availability* SW/HW support required

State-of-Art Solutions

SoA techniques leverage System Management Mode to address forensic requirements.

State-of-Art Solutions

SoA techniques leverage System Management Mode to address forensic requirements.

Issues

✓*atomicity*	CPU is *frozen*
✗*reliability*	integrity guarantee not provided
✗*availability*	prior-installation hardware required (i.e., PCI)
	no attempt to read more than 4GB

State-of-Art Solutions

SoA techniques leverage System Management Mode to address forensic requirements.

Issues

✓*atomicity*	CPU is *frozen*
✗*reliability*	integrity guarantee not provided
✗*availability*	prior-installation hardware required (i.e., PCI)
	no attempt to read more than 4GB

We can do better!

A step back

What's SMM?

System Management Mode is a mode of operation (similar to real mode) of Intel CPUs designed to handle system-wide functionality (e.g., power management and hardware control).

- code executed in an *isolated processor environment*
- *transparent to the OS*
- mode of operation with the *greatest level of privilege* (ring -2)
- address and operand size override prefixes allow 32bit data access

A step back

What's SMM?

System Management Mode is a mode of operation (similar to real mode) of Intel CPUs designed to handle system-wide functionality (e.g., power management and hardware control).

- code executed in an *isolated processor environment*
- *transparent to the OS*
- mode of operation with the *greatest level of privilege* (ring -2)
- address and operand size override prefixes allow 32bit data access

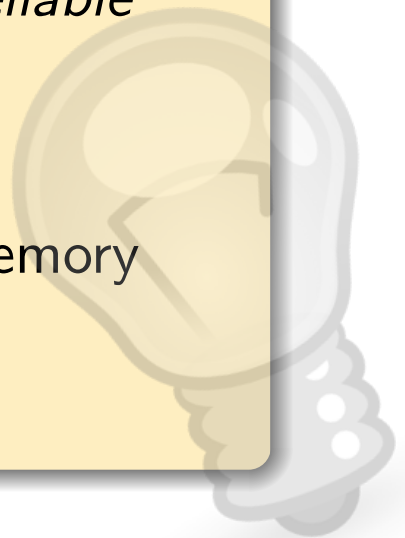
Critical Issue

SMM can access at most 4GB of physical memory!

SMMDumper

Contributions

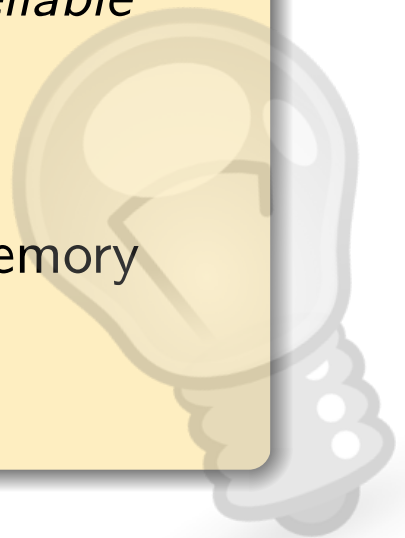
1. firmware-based technique to *atomically* perform a *reliable* memory dump (IA32)
2. dump physical memory *exceeding 4GB* (PAE)
3. *integrity guarantee* provided by signing the whole memory dump
4. QEMU-based prototype implemented



SMMDumper

Contributions

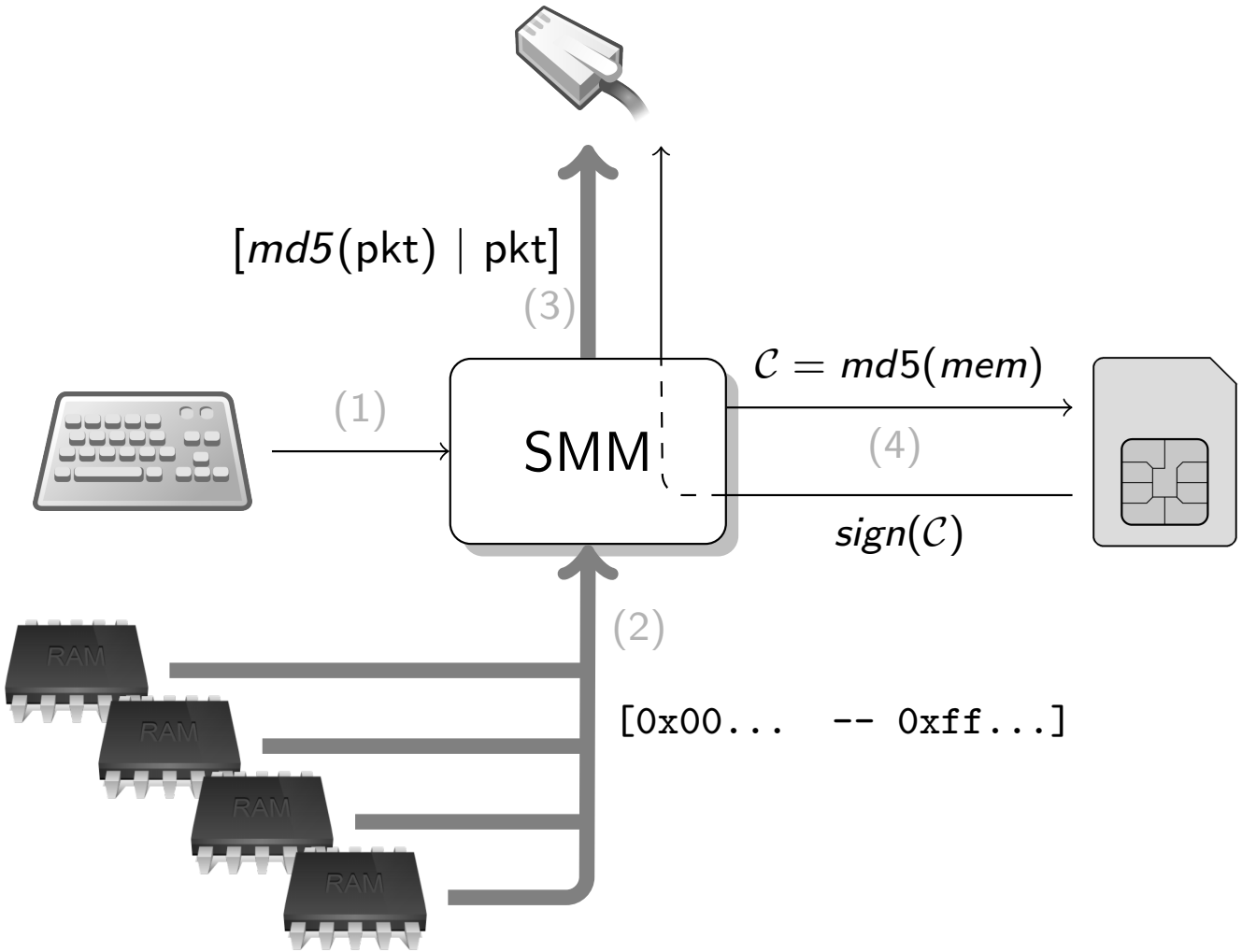
1. firmware-based technique to *atomically* perform a *reliable* memory dump (IA32)
2. dump physical memory *exceeding 4GB* (PAE)
3. *integrity guarantee* provided by signing the whole memory dump
4. QEMU-based prototype implemented



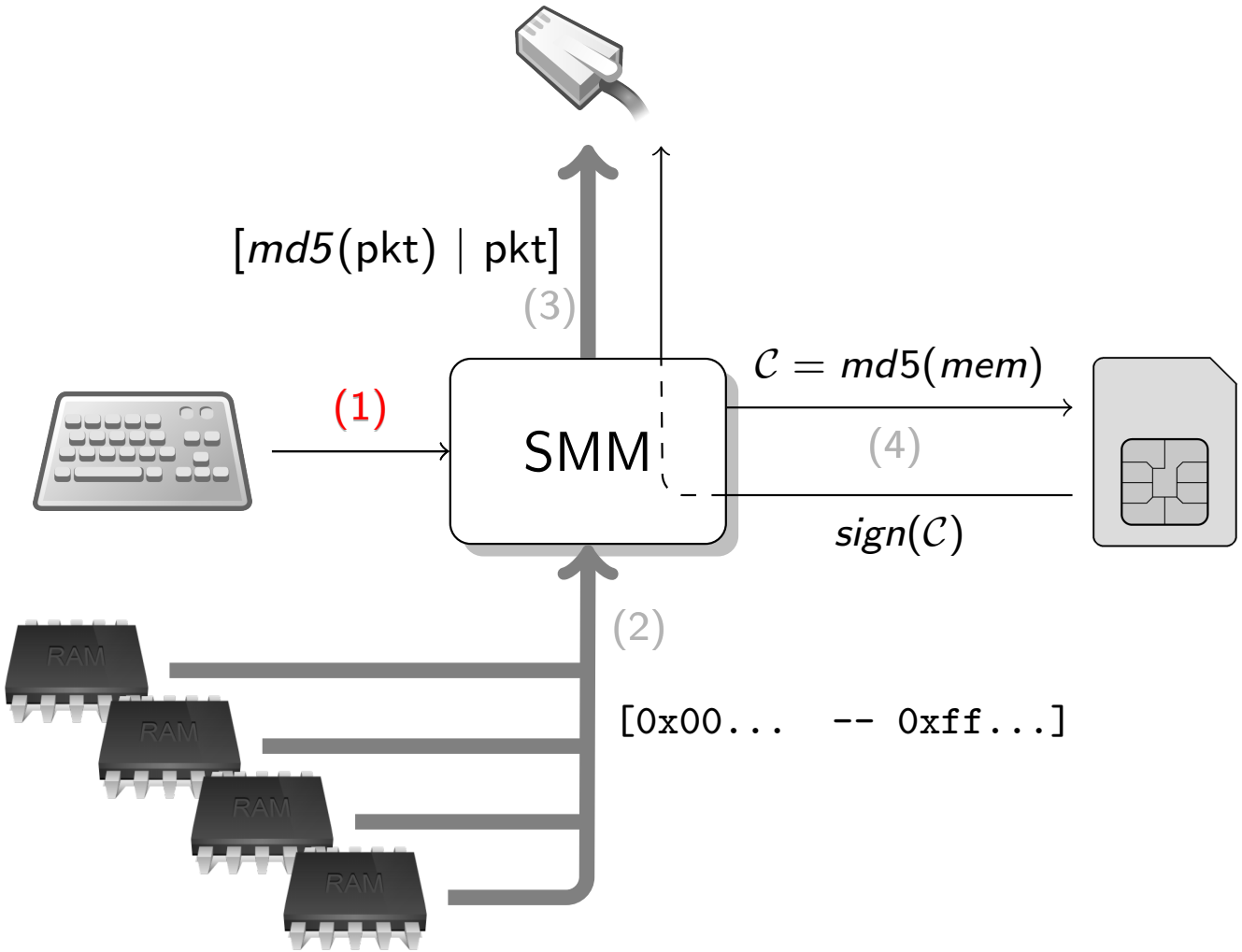
Threat Model

- the attacker has root access to the compromised system
- the attacker has compromised other machines in the same LAN
- the attacker can perform network attacks
- the attacker can **NOT** install an HW hypervisor

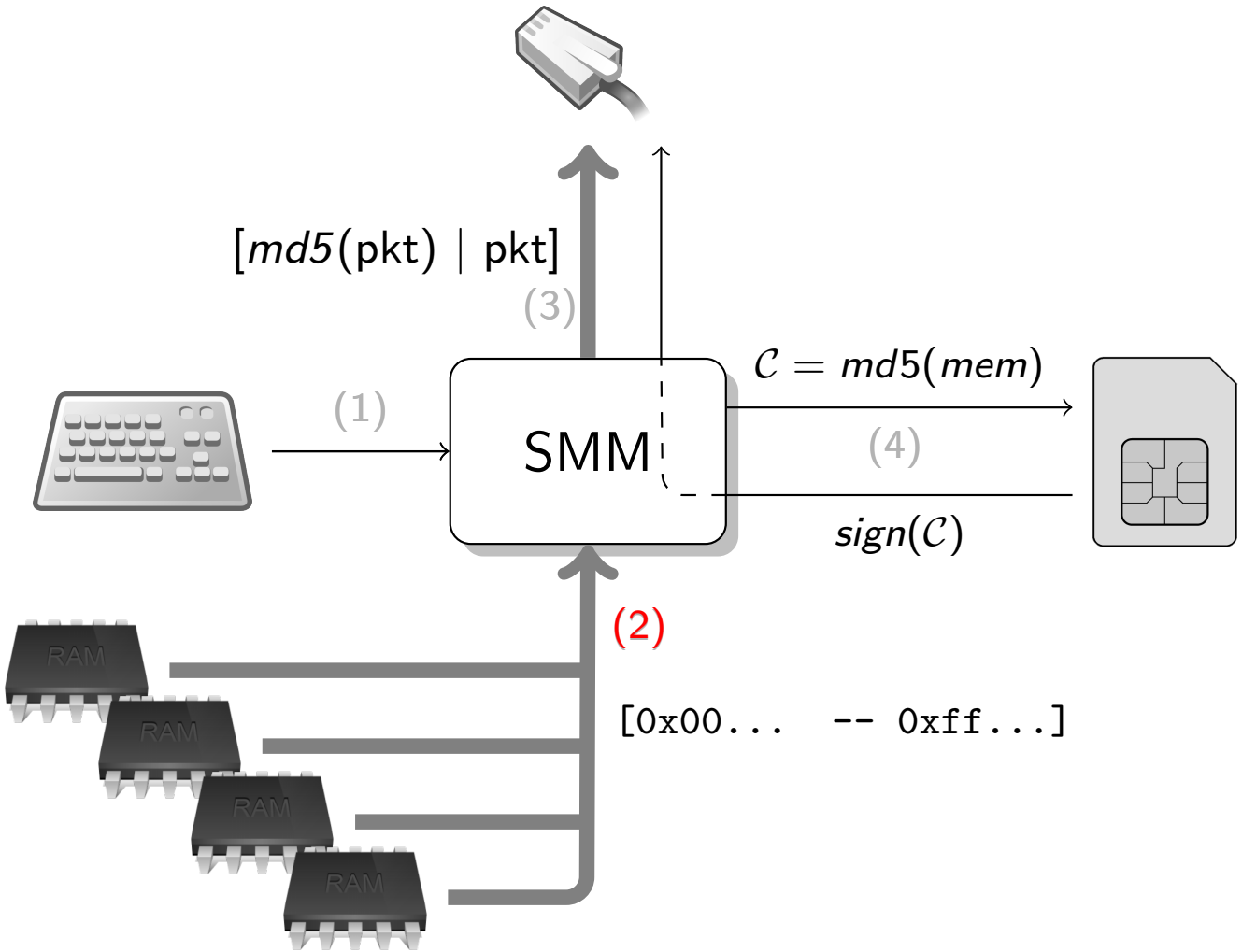
Overview of SMMDumper



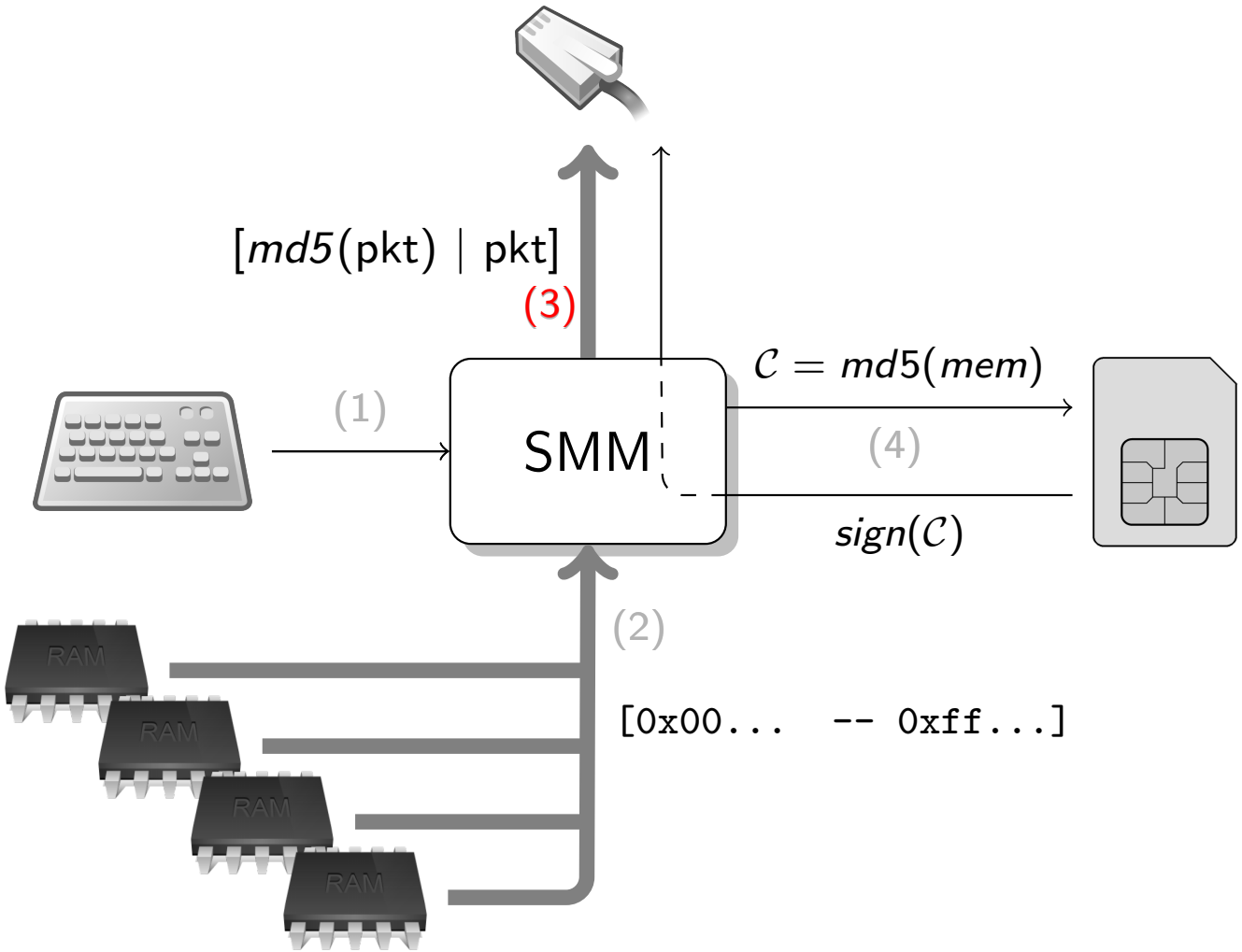
Overview of SMMDumper



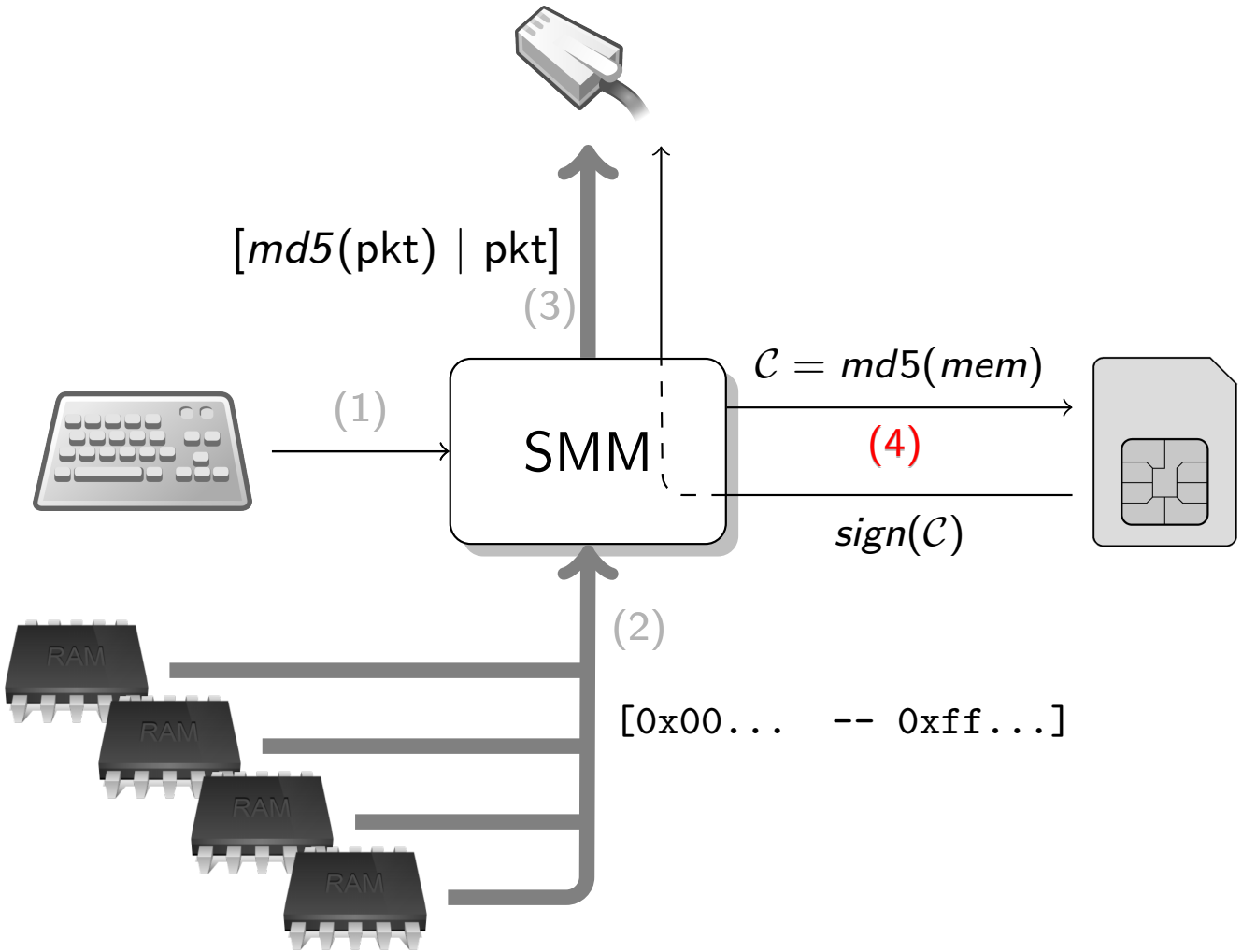
Overview of SMMDumper



Overview of SMMDumper



Overview of SMMDumper



SMMDumper

Challenges

1. trigger `SMI` to switch to `SMM`
2. guarantee the integrity of the collected data on the host as well as while in transit to a generic device
3. access *all* physical memory (even if it is greater than 4GB in size)

SMMDumper: Triggering System Management Interrupt

System Management Interrupt

- external `SMM` interrupt pin (`SMI#`)
- Advanced Programmable Interrupt Controller (`APIC`)

Bulletproof Triggering Implementation

hardware-based activation mechanisms:

- i.e., specific keystroke connected to the SMI pin (i.e. events specified by the I/O Controller Hub)
- isolate the SW component from user- and kernel-space

Software Triggering Implementation

SMM keylogger:

- `I/O APIC` contains a *Redirection Table* which routes `EXTINT`s to the CPUs
- *Redirection Table* is set to deliver `SMI` when `IRQ1` is asserted
- the keyboard scancode is read from the keyboard controller buffer
- an `IPI` message is sent to delivery the `IRQ1` to the CPU as soon as `rsm` is executed

SMMDumper: Data Integrity and Transmission

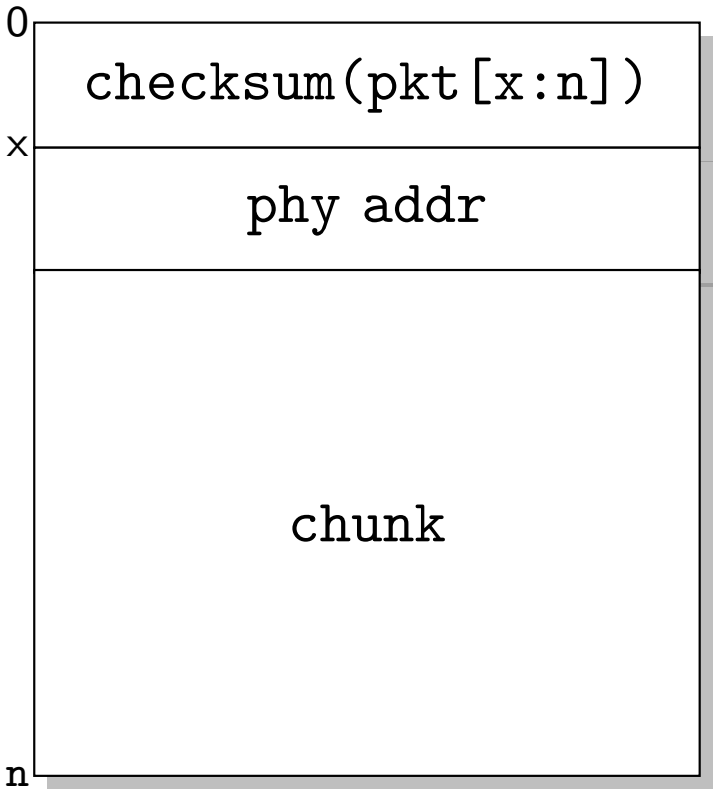
Network Transmission/Retransmission

- simple network SMM driver (polling mode)
- UDP protocol
- retransmission of lost or corrupted data supported

SMMDumper: Data Integrity and Transmission

Network Transmission/Retransmission

- simple network SMM driver (polling mode)
- UDP protocol
- retransmission of lost or corrupted data supported



Communication Protocol

- data divided in `chunks` of fixed size
- `phy_addr` used by the receiver to handle out-of-order or missing chunks
- `checksum` over the packet payload

Signing the Whole Memory Dump

1. as soon as SMMDumper starts, a smart card device D is plugged in
2. an incremental checksum C (`MD5`) of the whole memory is computed
3. once the memory dump is completed, C is sent to a smart card device D
4. D signs C with the private key stored inside the smart card
5. the receiver verifies the signature and compares C against the gathered memory dump

SMMDumper: Accessing Physical Memory

SMM limitations

- `SMM` similar to real mode
- override prefixes used to access up to 4GB
- paging disabled
- physical direct memory access

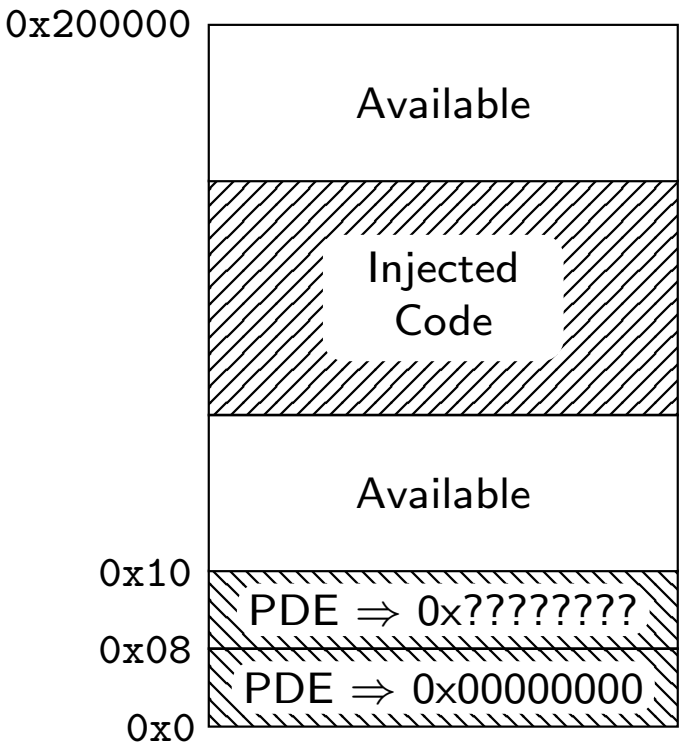
SMMDumper: Accessing Physical Memory

SMM limitations

- `SMM` similar to real mode
- override prefixes used to access up to 4GB
- paging disabled
- physical direct memory access

. . . but, still, **how can we read more than 4GB?**

SMMDumper: Accessing more than 4GB (IA32 - PAE)




```
1   va       = 0x00200000
2   p_pde    = 0x00000008
3   phy_addr = 0x100000000 /* 36-bit */
4   while phy_addr < MAX_MEMORY
5     /* Setup PDE */
6     p_pde->page_base_addr = phy_addr
7     p_pde->p  = 1 /* Present */
8     p_pre->us = 1 /* User/Super */
9     /* Now 0x00200000 points to phy_addr */
10    offset = 0
11    while offset < PAGE_SIZE:
12      packet =  str(phy_addr+offset)
13      packet += va[offset:offset+CHUNK_SIZE]
14      packet += MD5(packet[0:len(packet)])
15      /* Send pkt */
16      /* Update overall checksum */
17      offset += CHUNK_SIZE
18    phy_addr += PAGE_SIZE
```

SMMDumper: Experimental Evaluation

Setup

- prototype based on coreboot (opensource BIOS)
- entirely coded in assembly (∼500LoC, 47% MD5 implementation)
- run on QEMU 1.0.1 (Intel 3GHz, 6GB RAM, 100Mbit)

SMMDumper: Experimental Evaluation

Setup

- prototype based on coreboot (opensource BIOS)
- entirely coded in assembly (∼500LoC, 47% MD5 implementation)
- run on QEMU 1.0.1 (Intel 3GHz, 6GB RAM, 100Mbit)

Data Transmission

- UDP packet = 1024(`chunk`) + 16(`MD5`) + 8(`phy_addr`)
- transfer time for 6GB ≈ 13.5min
- ≈ 10% time overhead due to `MD5` calculation
- 144MB of metadata

SMMDumper: Experimental Evaluation

Atomicity

- QEMU instrumented to take a snapshot of the whole physical memory before starting to execute SMMDumper
- comparison of such a dump with the one of SMMDumper demonstrates accuracy and consistency
- some changes may occur when reading I/O memory mapped regions
  - these changes do not violate *atomicity* as not relevant for the analysis

Reliability

- simulation of man-in-the-middle attack
- payload modified and checksum updated
- result: receiver detected that the signature was invalid

SMMDumper: Conclusion

When Hardware Meets Software: A Bulletproof Solution to Forensic Memory Acquisition

SMMDumper

- SMMDumper is able to dump more than 4GB of memory
- SMMDumper satisfies the forensic requirements:
  - ✓*atomicity* no changes occurred to memory content
  - ✓*reliability* integrity is guaranteed
  - ✓*availability* completely OS and device independent

SMMDumper: Conclusion

When Hardware Meets Software: A Bulletproof Solution to Forensic Memory Acquisition

SMMDumper

- SMMDumper is able to dump more than 4GB of memory
- SMMDumper satisfies the forensic requirements:
  - ✓*atomicity* no changes occurred to memory content
  - ✓*reliability* integrity is guaranteed
  - ✓*availability* completely OS and device independent

Future Work

- extension to Intel 64bit CPU (*under submission*)
- support for multiprocessor system

Thanks for your attention!

Questions?

Alessandro Reina
alessandro.reina@unimi.it