Acquisition and Analysis of Compromised Firmware Using Memory Forensics

By

Johannes Stuttgen, Stefan Voemel and Michael Denzel

Presented At

The Digital Forensic Research Conference

DFRWS 2015 EU Dublin, Ireland (Mar 23rd- 26th)

DFRWS is dedicated to the sharing of knowledge and ideas about digital forensics research. Ever since it organized the first open workshop devoted to digital forensics in 2001, DFRWS continues to bring academics and practitioners together in an informal environment. As a non-profit, volunteer organization, DFRWS sponsors technical working groups, annual conferences and challenges to help drive the direction of research and development.

http:/dfrws.org

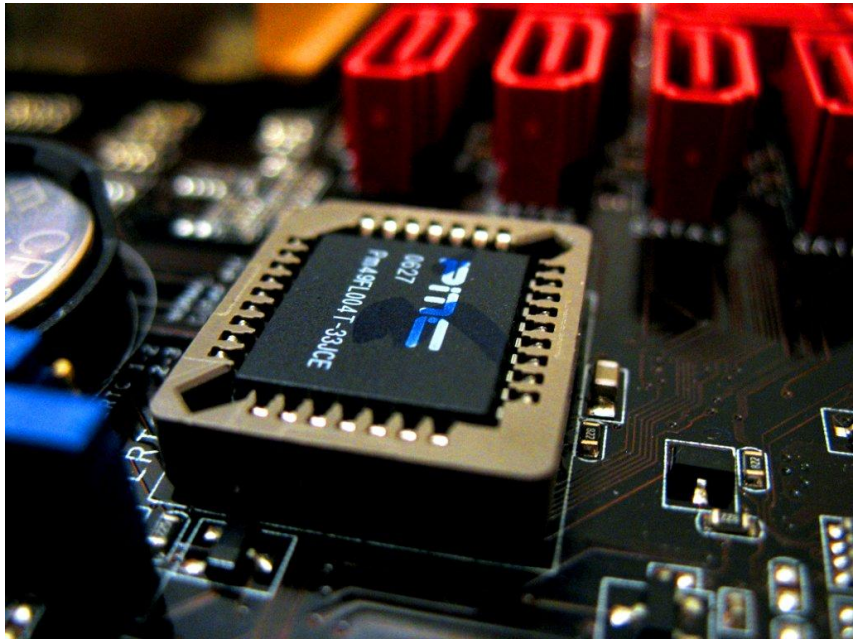
Acquisition and Analysis of Compromised Firmware Using Memory Forensics

Johannes Stüttgen, Stefan Voemel and Michael Denzel

Problem Statement

Discover and acquire firmware code on x86 systems using memory forensics:

- Enumerate firmware memory in the physical address space
- Acquire firmware code and data via the memory bus



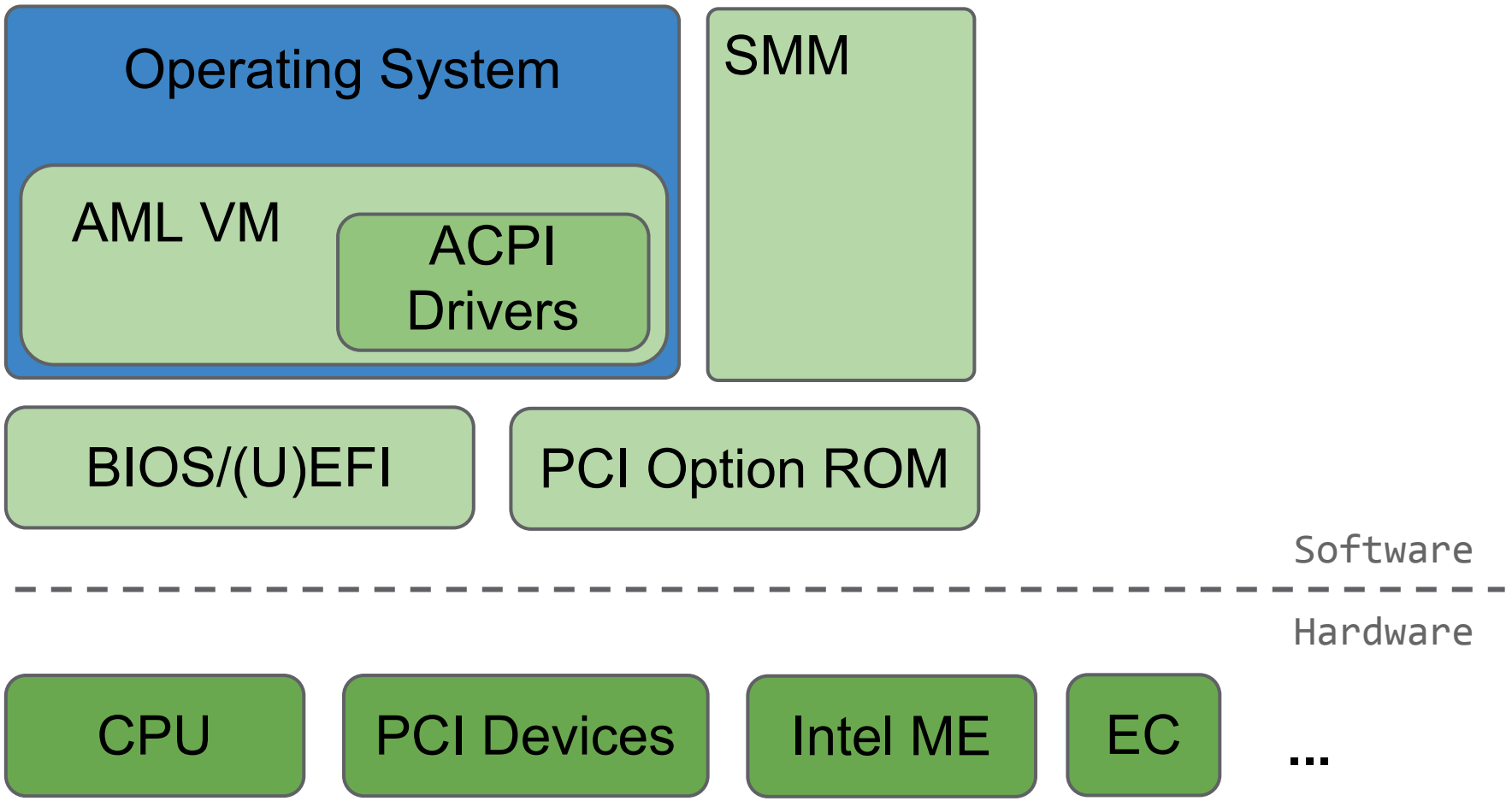
Related: Butterworth John, Xeno Kovah. *"Copernicus 2: SENTER the dragon"* CanSecWest (2014)

- Use Intel TXT and TPM to acquire firmware ROM
- Read firmware directly from SPI bus

Outline

- X86 Firmware Technologies
- Malicious Firmware Techniques
  - BIOS/(U)EFI
  - PCI Option ROM
  - ACPI
- Acquiring Firmware Code Using Memory Forensics
  - Locating Firmware in Memory
  - Accessing Firmware Memory
- Firmware Analysis
- Challenges

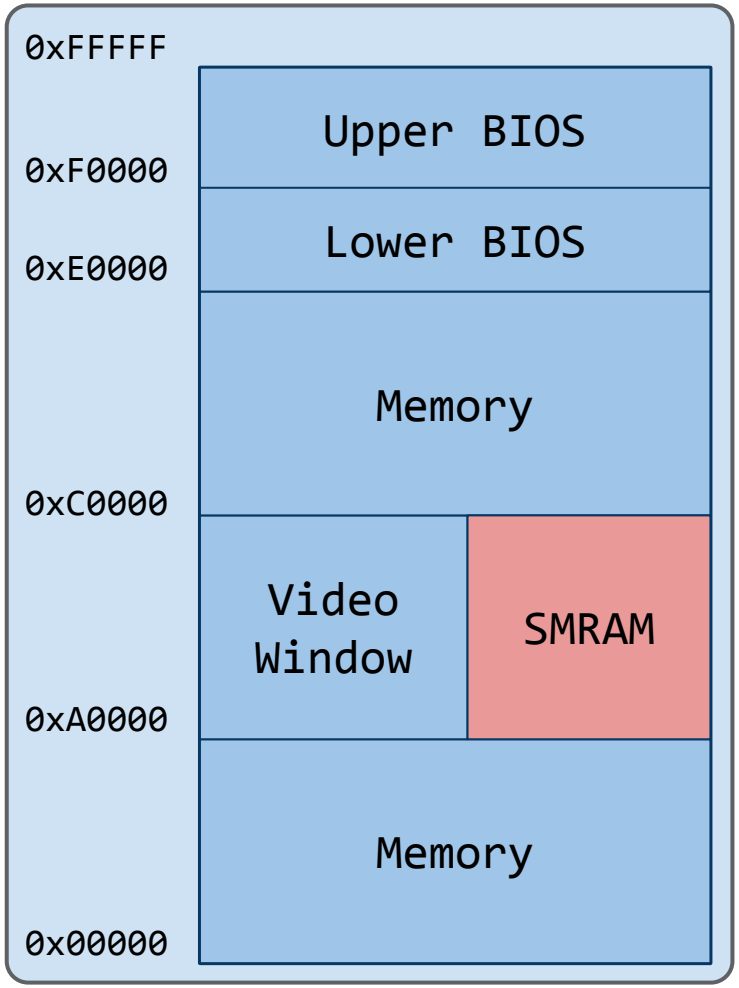
X86 Firmware Stack



BIOS / (U)EFI

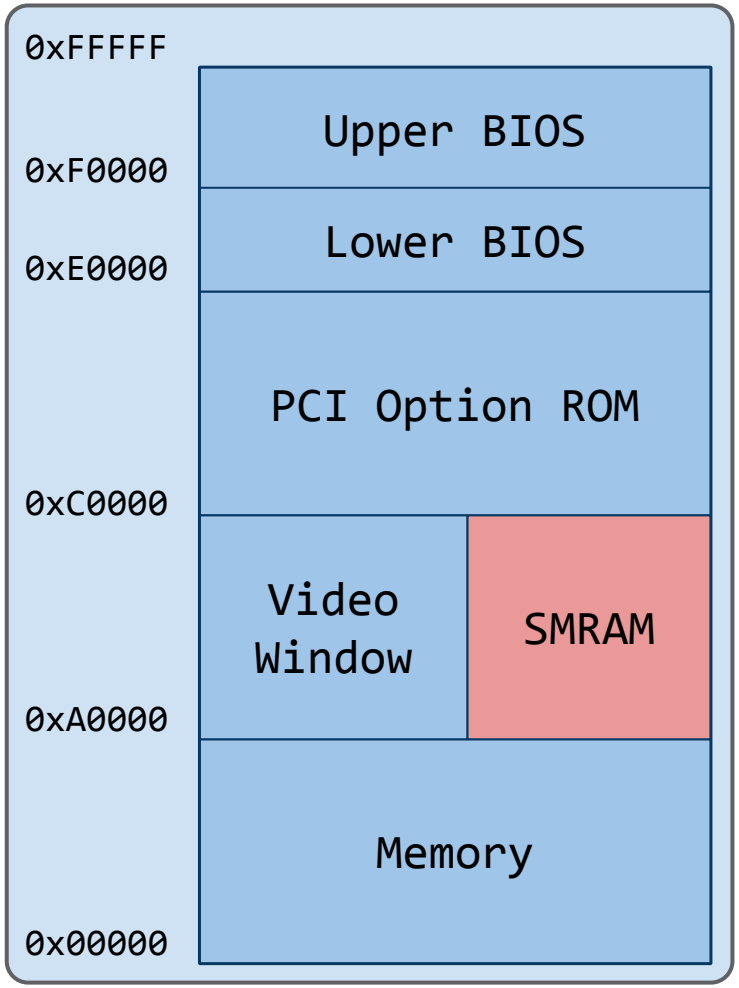
- BIOS code is stored on ROM chip
  - connected to southbridge via SPI
  - Mapped to `0xF0000 - 0xFFFFF` and `0xFFFF0000 - 0xFFFFFFFF`
- x86 CPU will load first instruction from `0xFFFFFFF0` when reset line is de-asserted (Reset Vector)
- Firmware initializes memory and migrates into RAM by modifying Programmable Attribute Map (PAM) registers

Note: (U)EFI has a different execution environment and driver model, the basic properties remain the same (Reset Vector, Mapped ROM, Migration to RAM)



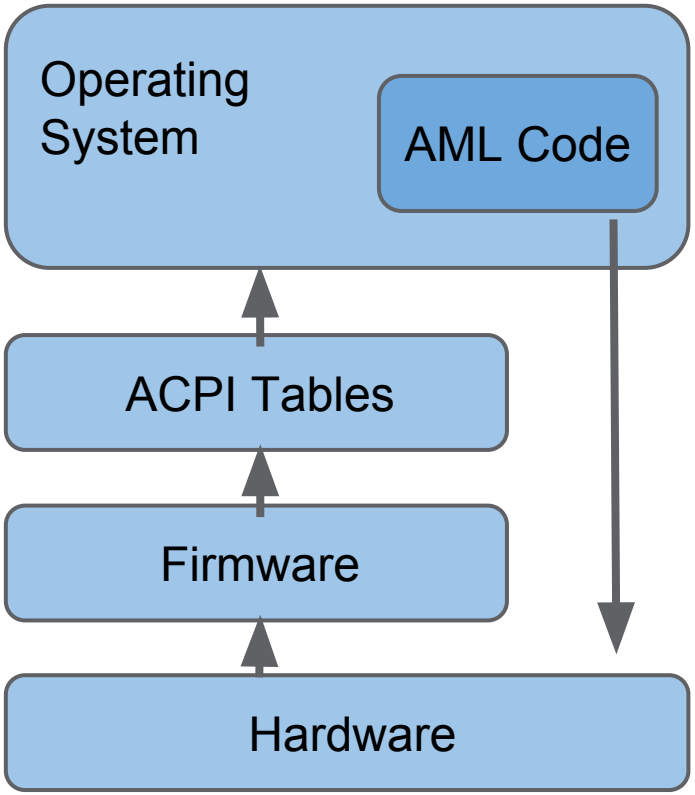
PCI Option ROM

- PCI devices require device specific initialization unknown to the firmware
- Initialization code stored in option ROM on the device
- Firmware maps ROM into physical address space and stores address of mapping in `XROMBAR` register in the devices PCI configuration space
- Firmware copies code from ROM into RAM (`0xC0000 - 0xE0000`)
- Initialization code is executed in firmware context with full privileges



ACPI

- ACPI Provides an abstract interface to platform configuration and power management
- ACPI code is written in ACPI Source Language (ASL) and compiled to bytecode called ACPI Machine Language (AML)
- Operating systems provide an implementation of an AML virtual machine
- Firmware provides information on ACPI registers and AML code in ACPI tables
- AML code runs in OS context



BIOS/EFI Rootkits

- Many BIOS implementations do not enforce code signing
- Already BIOS malware in the wild (Mebromi, 2011)
- (U)EFI and some BIOS implementations require signed updates
- ROM chip is replaceable/flashable with physical access
  - Brossard, Jonathan. *"Hardware Backdooring is practical."* BlackHat, Las Vegas*, USA* (2012).
- Signature implementations have bugs
  - Kallenberg, Corey, et al. *"Extreme Privilege Escalation On Windows 8/UEFI Systems."* Black Hat (2014).
- CRTM secure boot with TPM is flawed
  - Butterworth, John, et al. *"Bios chronomancy: Fixing the core root of trust for measurement."* ACM SIGSAC (2013).
  - Trust firmware to submit measurements to TPM
  - Firmware can be patched to still submit correct measurements
- Firmware updates are frequently ignored, so vulnerabilities remain unpatched for years
  - Corey Kallenberg, et al. *“How Many Million BIOSes Would you Like to Infect?”* CanSecWest 2015

PCI Option ROM Rootkits

- Patch option ROM on existing devices
  - Heasman, J. *"Implementing and detecting a PCI rootkit."* Black Hat USA (2006)
- Or connect a new device with a malicious option ROM
  - Loukas, K. *"De Mysteriis Dom Jobsivs–Mac EFI Rootkits."* Black Hat USA (2012)

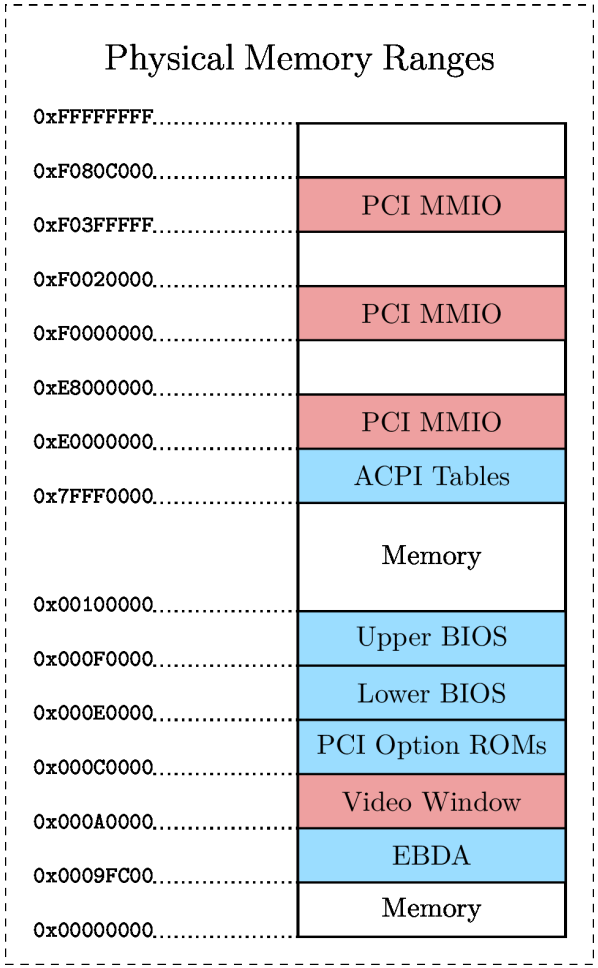
picture: http://ho.ax/De_Mysteriis_Dom_Jobsivs_Black_Hat_Paper.pdf

ACPI Rootkits

- BIOS copies ACPI code into RAM
- OS scans for “magic” signature
- OS executes AML code in AML VM in kernel mode
- Attacker can:
  - Patch BIOS to provide malicious ACPI tables
  - Put malicious ACPI code in memory in front of benign ACPI tables, because the OS will use first tables it can find
- ACPI code can read/write arbitrary kernel memory
- Heasman, John. *"Implementing and detecting an ACPI bios rootkit."* Black Hat Europe (2006).

Firmware Acquisition Using Memory Forensics

- All relevant firmware code and data regions are memory resident at runtime
- We can expand the memory acquisition process to acquire not only OS memory, but firmware memory as well
- We need to solve two problems:
  - Make sure not to access MMIO regions, as this can destabilize the system.
  - Avoid kernel APIs when mapping memory, as they were not intended for this and can fail.



Scope of Current Acquisition Tools

- Current tools only acquire memory marked as RAM by the OS
  - Prevent problems from accidentally reading device memory
- Major OS like Windows, Linux and OSX don't consider firmware regions to be RAM

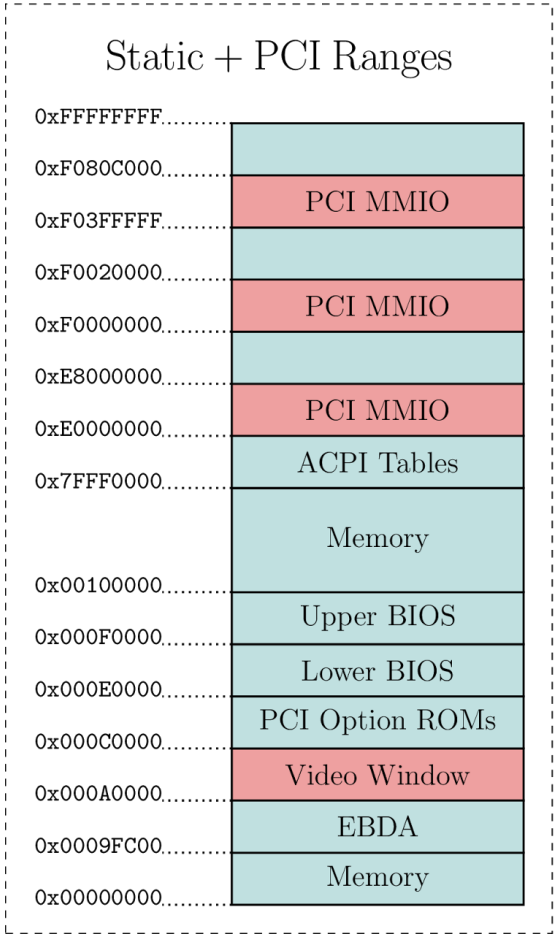
```
#define LIME_RAMSTR "System RAM"
...
for (p = iomem_resource.child; p ; p = p->sibling) {
                if (strcmp(p->name, LIME_RAMSTR))
                        continue;
                ...
}
```

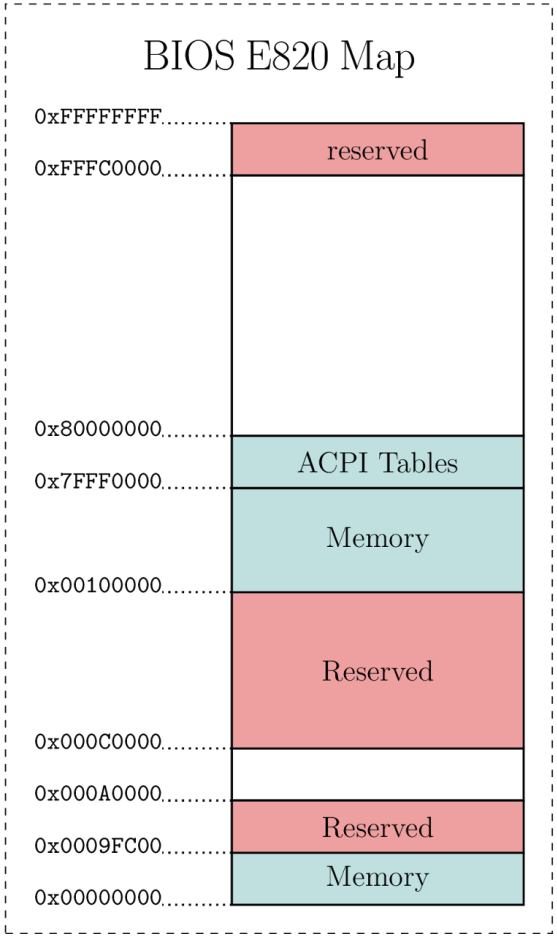
```
00000000-00000fff : reserved
00001000-0008efff : System RAM
0008f000-0008ffff : ACPI NV
00090000-0009ffff : System RAM
000a0000-000bffff : PCI Bus 00
000c0000-000c7fff : Video ROM
000c8000-000cbfff : pnp 00:00
000cc000-000cffff : pnp 00:00
000d0000-000d3fff : pnp 00:00
000d4000-000d7fff : pnp 00:00
000d8000-000dbfff : pnp 00:00
000dc000-000dffff : pnp 00:00
000e0000-000e3fff : pnp 00:00
000e4000-000e7fff : pnp 00:00
000e8000-000ebfff : pnp 00:00
000ec000-000effff : pnp 00:00
000f0000-000fffff : System ROM
00100000-da99efff : System RAM
...
```

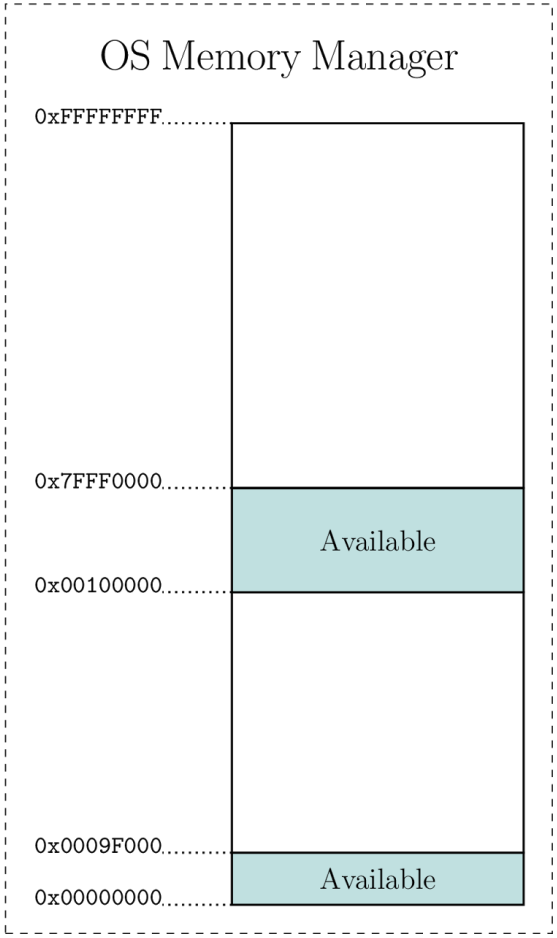
Enumerating Memory with PCI Introspection

- PCI Devices implement a configuration data structure
- It can be accessed via I/O ports `CONFIG_ADDRESS` (`0xCF8`) and `CONFIG_DATA` (`0xCFC`)
- Firmware configures MMIO for the device by writing the address of mapped regions to the BARs in configuration space for a device
- We can enumerate all MMIO regions by parsing the configuration space of all PCI devices
- We can safely read all memory (including firmware) by just skipping these regions

Completing the View on the Physical Address Space

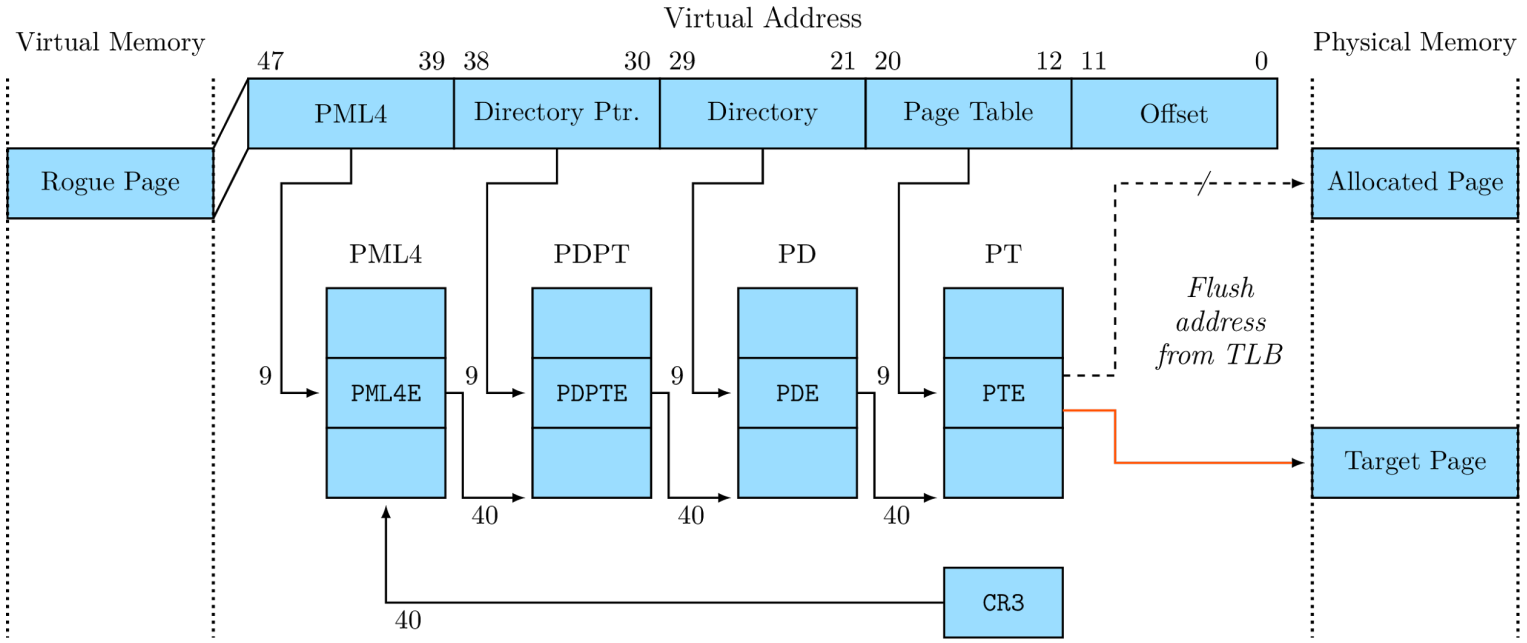






Mapping Firmware Memory

- We avoid OS APIs to prevent failure due to multiple mappings with different cache attributes
- Remap page in drivers data segment manually by editing the page tables and then flushing the page from the TLB (`invlpg`)



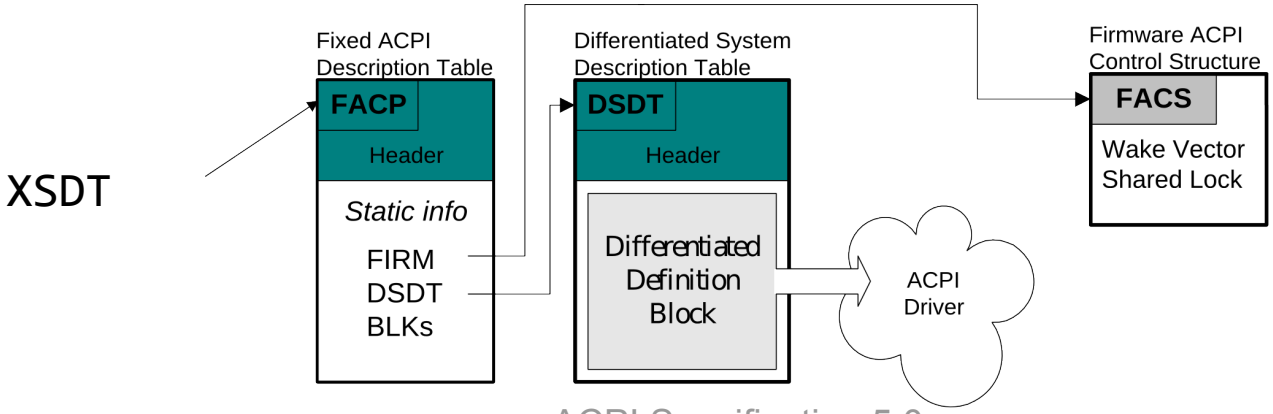
Implementation

- Technique is already implemented in Winpmem since version 1.5.2
- We have also ported it to Linux, not released yet
- In our evaluation, these two implementations were the only ones who could acquire firmware memory without crashing the system

Acquisition Tool	Firmware Acquired
Memoryze	✗
FTK Imager	✗
Moonsols DumpIt	✗
WinPmem	✗
WinPmem (pci)	✓
WindowsMemoryReader	✗
LiMe	✗
Pmem	✗
Pmem (pci)	✓

Firmware Analysis

- BIOS, (U)EFI and PCI Option ROMs can be dumped from the memory image for analysis with specialized software, as they reside in well defined regions in the physical address space
- The location of ACPI tables is not defined
  - OS finds them by scanning for a signature
- We have replicated this algorithm in a Volatility plugin
  - Dumps all ACPI tables to the file system



source: ACPI Specification 5.0

Operation Regions

- AML uses a construct called *Operating Regions* to access physical memory
- We have created an ACPI table scanner which analyzes all operating regions and flags references to kernel memory
- The scanner helps to focus on analyzing the relevant parts of the code

	Correctly Classified	Falsely Classified	$\sum$
Malicious	13.0%	16.4%	29.4%
Benign	61.9%		61.9%
Unknown	8.7%		8.7%
$\sum$	83.6%	16.4%	100%

Challenges

- SMRAM is only accessible in SMM
  - Restriction is enforced by memory controller
  - Set up by BIOS/(U)EFI before boot
  - Embleton, Shawn, Sherri Sparks, and Cliff C. Zou. *"SMM rootkit: a new breed of OS independent malware."* Security and Communication Networks 6.12 (2013): 1590-1605.
- Management Engine runs on separate CPU (ARC4, 32-bit RISC)
  - Full RAM access
  - Dedicated interrupt line to NIC
  - Dedicated RAM inacessible by CPU (ME-UMA)
  - Tereshkin, Alexander, and Rafal Wojtczuk. *"Introducing ring-3 rootkits."* Black Hat USA (2009).
- Embedded Controller exists in many modern laptops
  - Separate CPU
  - Separate Memory
  - Runs even when CPU is turned off
  - Weinmann, Ralf-Philipp. *“The Hidden Nemesis: Backdooring Embedded Controllers”* Chaos Communication Congress (2010)

Conclusion

- Malicious firmware is a real threat
- There are many memory regions containing firmware artifacts
  - Ignored by most current memory forensic tools
  - Can be acquired using PCI and MMU assisted methods
- We developed tools to acquire firmware on Windows and Linux
  - Extract firmware from memory images
  - Find indicators of compromise in ACPI tables

Questions?

Contact: johannes.stuettgen@gmail.com