Anti-Forensic Resilient Memory Acquisition

By

Johannes Stuttgen and Michael Cohen

Presented At

The Digital Forensic Research Conference

DFRWS 2013 USA Monterey, CA (Aug 4th - 7th)

Anti-Forensic Resilient Memory Acquisition

Johannes Stüttgen[+] and Michael Cohen[*]

[+]Department of Computer Science
University Erlangen-Nuremberg
Germany

[*]Google Inc.
Zurich
Switzerland

07.08.2013

Motivation

- Physical memory analysis increasingly common in IR scenarios
- Memory images are often acquired by software
- This has raised the attention of malware authors
- We analyse the attack surface of current acquisition tools and propose an anti-forensic resilient approach for memory acquisition

Scenario

- Live Analysis
- Compromised Machine
- No physical access
    - Software Acquisition
- No access to hardware virtualization support
- Ability to load driver/kernel module

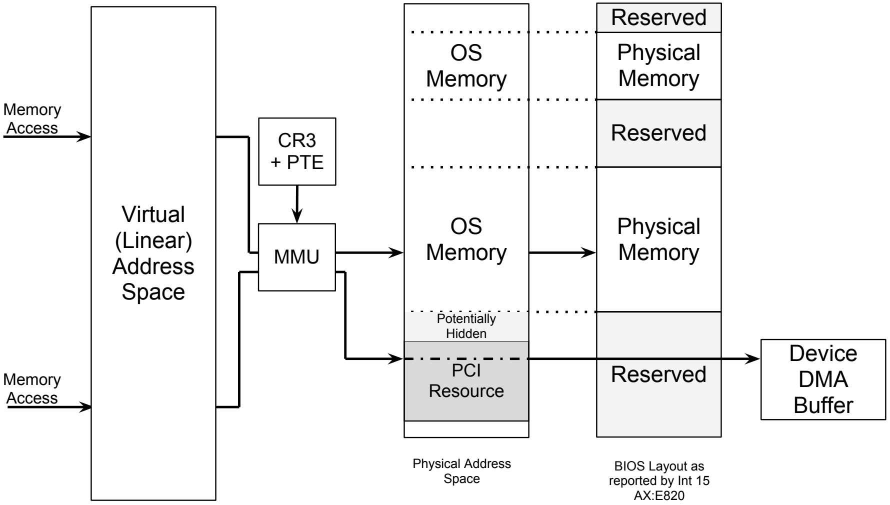
Motivation

- Physical memory analysis increasingly common in IR scenarios
- Memory images are often acquired by software
- This has raised the attention of malware authors
- We analyse the attack surface of current acquisition tools and propose an anti-forensic resilient approach for memory acquisition

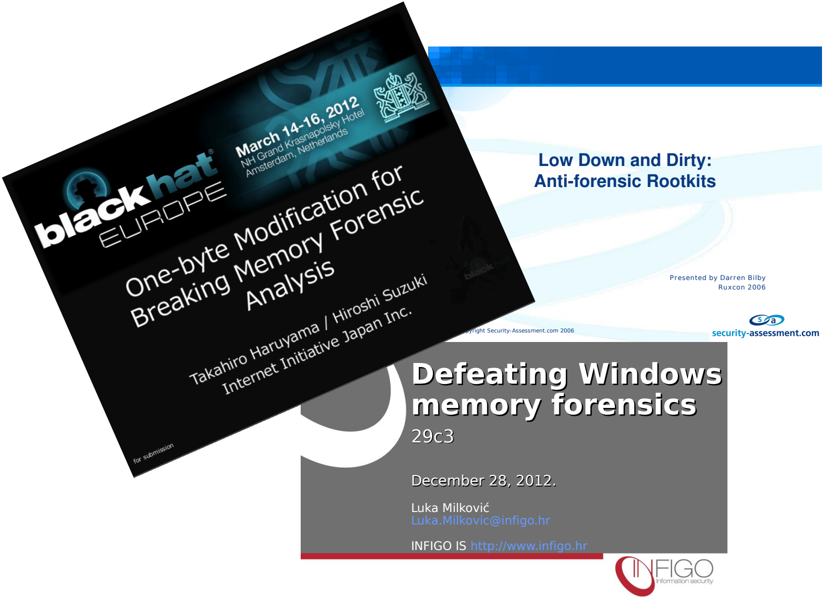
Scenario

- Live Analysis
- Compromised Machine
- No physical access
  - Software Acquisition
- No access to hardware virtualization support
- Ability to load driver/kernel module

Accessing the Physical Address Space



The Anti-Forensics Problem



The Live Analysis Dilemma

Inherent Problems of Live Forensics

- We work on a potentially compromised machine
- Using a potentially subverted operating system
- With the same privileges as an intruder
- Who was there first

Conclusions

- Perform only the most essential steps of analysis on the system
- Using as little APIs as possible
- On the highest possible privilege level
- And still be aware our results might be wrong

The Live Analysis Dilemma

Inherent Problems of Live Forensics

- We work on a potentially compromised machine
- Using a potentially subverted operating system
- With the same privileges as an intruder
- Who was there first

Conclusions

- Perform only the most essential steps of analysis on the system
- Using as little APIs as possible
- On the highest possible privilege level
- And still be aware our results might be wrong

Experiment with Current Tools

- Take a „modern“ system (Win 7 x64 SP1)
- Manipulate a number of commonly used APIs and structures for
  - Memory Enumeration (`GetPhysicalMemoryRanges`)
  - Memory (`MapMemoryDumpMdl`)
  - Location of Kernel Symbols (`KDBG`)
- Evaluate the performance of „current“ memory acquisition tools

Tool	Version	Format	KDBG	GetPhysicalMemoryRanges	MapMemoryDumpMdl
Memoryze	2.0	`raw`	✓	✗	✓
FTK Imager	3.1.2	`raw`	✓	✗	✓
Win64dd	1.4.0	`raw`	✓/✗	✗	✗
Win64dd	1.4.0	`dmp`	✗	✗	✗
DumpIt	1.4.0	`raw`	✓	✗	✗
WinPmem	1.3.1	`raw`	✗	✗	✓
WinPmem	1.3.1	`dmp`	✗	✗	✓
WindowsMemoryReader	1.0	`raw`	✓	✗	✓
WindowsMemoryReader	1.0	`dmp`	✓	✗	✓

The Cause of these Problems

Trust compromised kernel

- To report the precise memory geometry
- To map memory correctly

Platform independent problem

- Windows
  - `MmGetPhysicalMemoryRanges()` / `SMBIOS`
  - `MapViewOfSection()` / `MmMapIoSpace()` / `MmMapMemoryDumpMdl()`
- Mac OS X
  - `PE_state.bootArgs`
  - `PhysicalMemoryDescriptor`
- Linux
  - `iomem_ressource`
  - `kmap()`

The Cause of these Problems

Trust compromised kernel

- To report the precise memory geometry
- To map memory correctly

Platform independent problem

- Windows
  - `MmGetPhysicalMemoryRanges()` / `SMBIOS`
  - `MapViewOfSection()` / `MmMapIoSpace()` / `MmMapMemoryDumpMdl()`
- Mac OS X
  - `PE_state.bootArgs`
  - `PhysicalMemoryDescriptor`
- Linux
  - `iomem_ressource`
  - `kmap()`

Possible Solution

Memory Enumeration

- Do not trust the information provided by BIOS/EFI/Kernel
- Find out where exactly MMIO is located
- Acquire everything except MMIO regions

Memory Mapping

- We don't need the kernel for that
- Any driver running in ring 0 has access to the page tables
- Edit the page tables ourselves to map physical memory
- Do so in a stealthy manner to make detection hard

Possible Solution

Memory Enumeration

- Do not trust the information provided by BIOS/EFI/Kernel
- Find out where exactly MMIO is located
- Acquire everything except MMIO regions

Memory Mapping

- We don't need the kernel for that
- Any driver running in ring 0 has access to the page tables
- Edit the page tables ourselves to map physical memory
- Do so in a stealthy manner to make detection hard

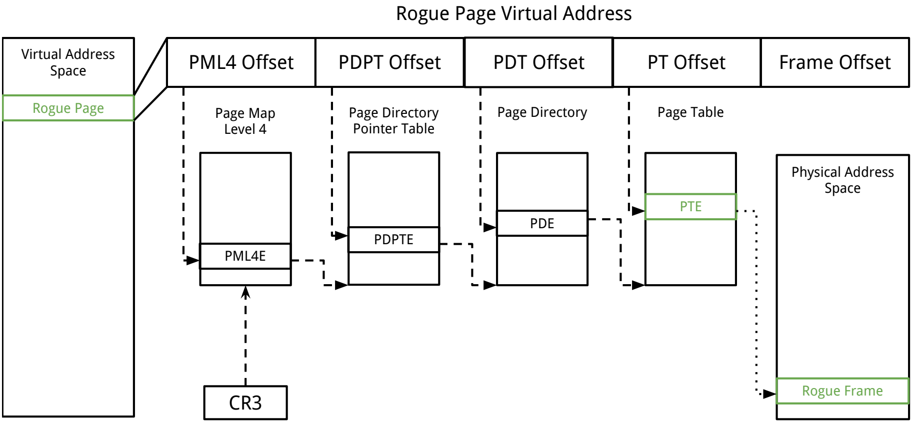
Memory Enumeration

- MMIO is managed by the northbridge
- Most devices are attached to PCI(e) bus
- Using Port I/O, we can query all PCI devices and bridges and retrieve the base address register (BAR) and buffer size

Feasability

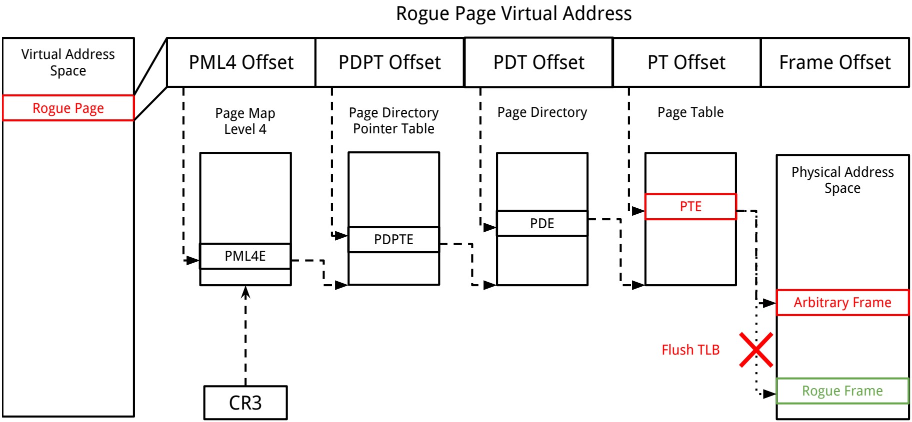
- This is exactly what `lspci` does
- It does so even from userspace
- `lspci -H1 -v`

Direct PTE Remapping I



- Allocate one non-paged page of memory
- Locate page table entry (PTE)

Direct PTE Remapping II



- Overwrite PTE physical offset
- Flush PTE from TLB

Stability

Concerns

- Software is not supposed to map physical page simultaneously with different cache attributes
- Operating system might use our PTE and crash
- There are more than just PCI devices connected to the memory bus (RCT, HPET, APIC, ...)

Experiences

- Since we only read from the rogue mapping caching shouldn't be a problem
- Using non-paged memory prevented the operating system from touching the rogue PTE in our tests
- For all standard devices present in our test system reading was not a problem

Limitations

Memory Enumeration

- A debug register hook can detect Port I/O
- With the general detect bit on this is virtually undetectable
- A rootkit could simulate a PCI device and mark its memory as device memory
- Of course this could also point an investigator directly to its code...

Memory Mapping

- A page-fault handler hook together with marking the page tables read only can detect direct modifications of the rogue PTE
- A solution to this could be creating and using our own page tables during the acquisition step

Limitations

Memory Enumeration

- A debug register hook can detect Port I/O
- With the general detect bit on this is virtually undetectable
- A rootkit could simulate a PCI device and mark its memory as device memory
- Of course this could also point an investigator directly to its code...

Memory Mapping

- A page-fault handler hook together with marking the page tables read only can detect direct modifications of the rogue PTE
- A solution to this could be creating and using our own page tables during the acquisition step

Conclusions

- Software acquisition can always be subverted on the same privilege level
- Currently it's shockingly simple, we should make this harder to do

Code Release

- Initial release for Windows
- Grab it at: `http://goo.gl/9VnnkY`
- Slides at `http://goo.gl/ALFfT4`
- Our code works on any x86 cpu, regardless of the OS
- Successful tests on OSX and Linux, expect cross platform release at `https://code.google.com/p/pmem`

Conclusions

- Software acquisition can always be subverted on the same privilege level
- Currently it's shockingly simple, we should make this harder to do

Code Release

- Initial release for Windows
- Grab it at: `http://goo.gl/9VnnkY`
- Slides at `http://goo.gl/ALFfT4`
- Our code works on any x86 cpu, regardless of the OS
- Successful tests on OSX and Linux, expect cross platform release at `https://code.google.com/p/pmem`

Tool Testing

- It is very important to test our tools to find such weaknesses
- Especially in the context of an active adversary
- We are not trying to criticise any specific tool
- The problem is a general one in any tool relying on kernel API's

Moonsols

- We would like to thank Mathieu Suiche for openly sharing his tools with us for testing
- He set a great example in a community where many others try to keep their methods secret

Tool Testing

- It is very important to test our tools to find such weaknesses
- Especially in the context of an active adversary
- We are not trying to criticise any specific tool
- The problem is a general one in any tool relying on kernel API's

Moonsols

- We would like to thank Mathieu Suiche for openly sharing his tools with us for testing
- He set a great example in a community where many others try to keep their methods secret

EOF

Questions?