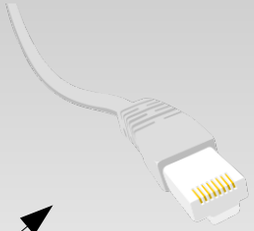
Memory Forensics

Davide Balzarotti
davide.balzarotti@eurecom.fr

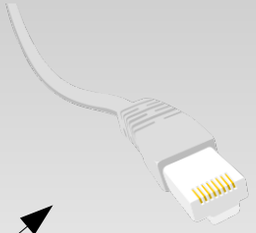
Previously, on Computer Forensics...



- Packets & Connections
- Services
- Transferred data

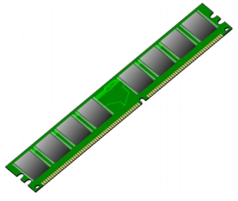
- Volumes and Partitions
- File systems
- OS & Apps Artifacts

Previously, on Computer Forensics...



- Packets & Connections
- Services
- Transferred data

- Volumes and Partitions
- File systems
- OS & Apps Artifacts



Memory

Memory Forensics

The process of capturing a copy of the system memory (RAM) and extracting a number of artifacts that are useful for an investigation

- Essentially, it consists in
  - Acquiring a snapshot of the system memory
  - Locating known data structures in the raw image to extract OS and process information
  - Carving de-allocated data structures, strings, encryption keys, credit card numbers, …
- Relatively new field (~2005), and still a very active research area

Memory Forensics - Pro

1. Memory is relatively small compared to hard disks
2. Attackers often overlook their memory footprint
3. Many of the artifacts used internally by the kernel can be used for forensics
4. Even rootkits designed to hide data in a running system need to be located somewhere in memory
5. Certain information (loaded kernel modules, open sockets, …) may be difficult to extract otherwise
6. Some malware samples only reside in memory

Memory Forensics - Cons

- The content of the memory keeps changing
  - Consecutive imaging acquisitions give different results
  - Forget about comparing MD5s

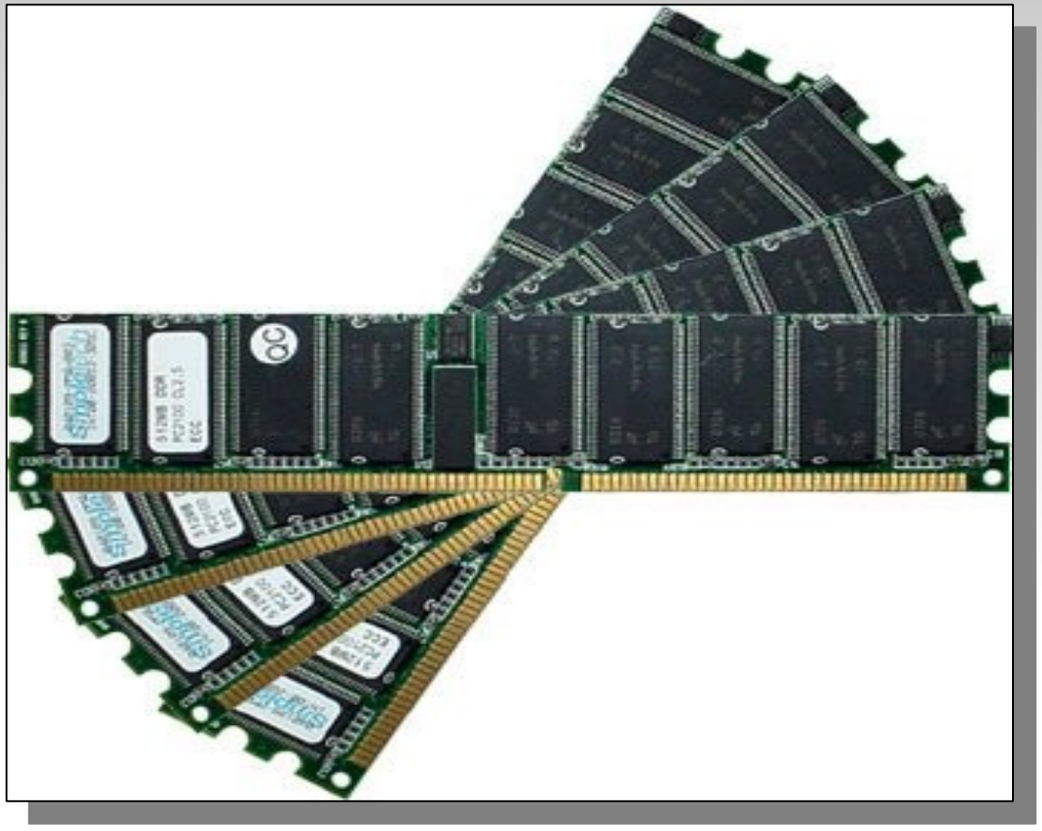
1. It is impossible to verify the authenticity of the acquired data

2. Data collection requires an efficient approach with a small footprint

- Data structures change among different OS versions
  - **Semantic Gap**: going from a raw sequence of bytes to high-level artifacts

Available Information

- Running, terminated, and hidden processes
  - Code, open files, buffers, ...
- Open sockets and active connections
- Memory-mapped files
  - Executables, shared libraries, kernel modules, …
- Clipboard's content
- Volatile registry hives
- Cached data
  - URLs, passwords, open documents, emails and IM messages, …



Memory Acquisition

Summary

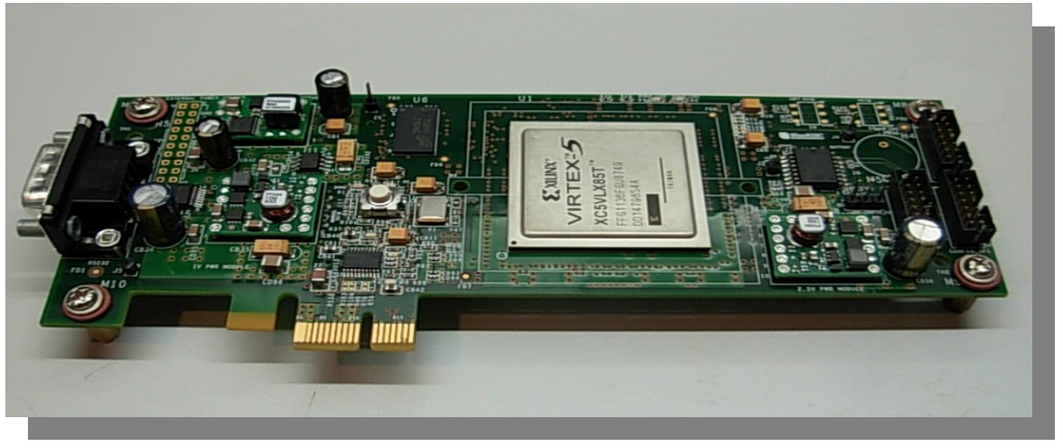
- IOMMU and Memory Layout
- Atomicity & memory footprint
- Software acquisition
- Hardware acquisition
- Cold-boot attack

Memory Acquisition

- Software Acquisition
  - Relies on a program running inside the system to read and store a copy of the memory
  - The software is altering the system, so its footprint is very important
- Hardware Acquisition
  - Relies on hardware devices to read the memory, often bypassing the CPU
  - It does not introduce any new artifact in the system
- Most of the existing approaches don't freeze the system during the acquisition, potentially leading to inconsistencies

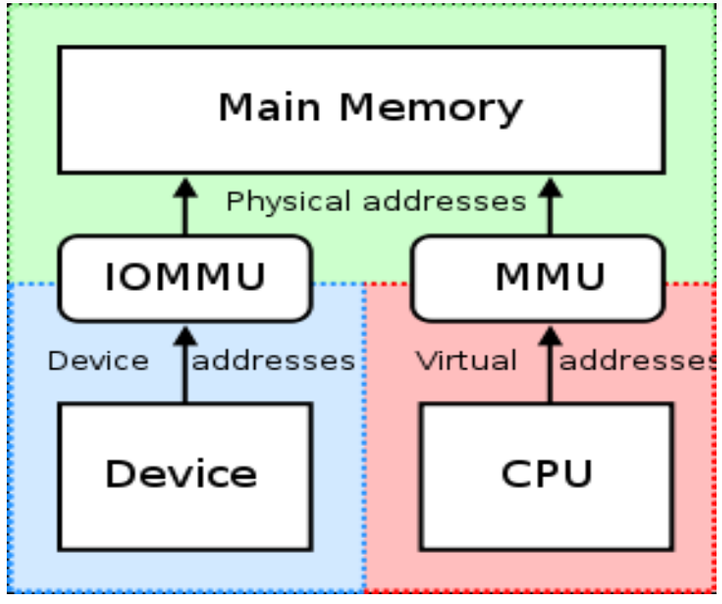
Hardware Acquisition

- Based on DMA transfer on an external bus
- Firewire
  - The Serial Bus Protocol 2 (SBP-2) specifies a method to directly access the main memory from a FireWire device
  - Tools exist (e.g., `1394memimage`) but..
  - Not forensically sound (it does not work with all systems and the target OS may crash)
- Internal acquisition cards
  - Need to be connected to the PCI bus *before* the incident

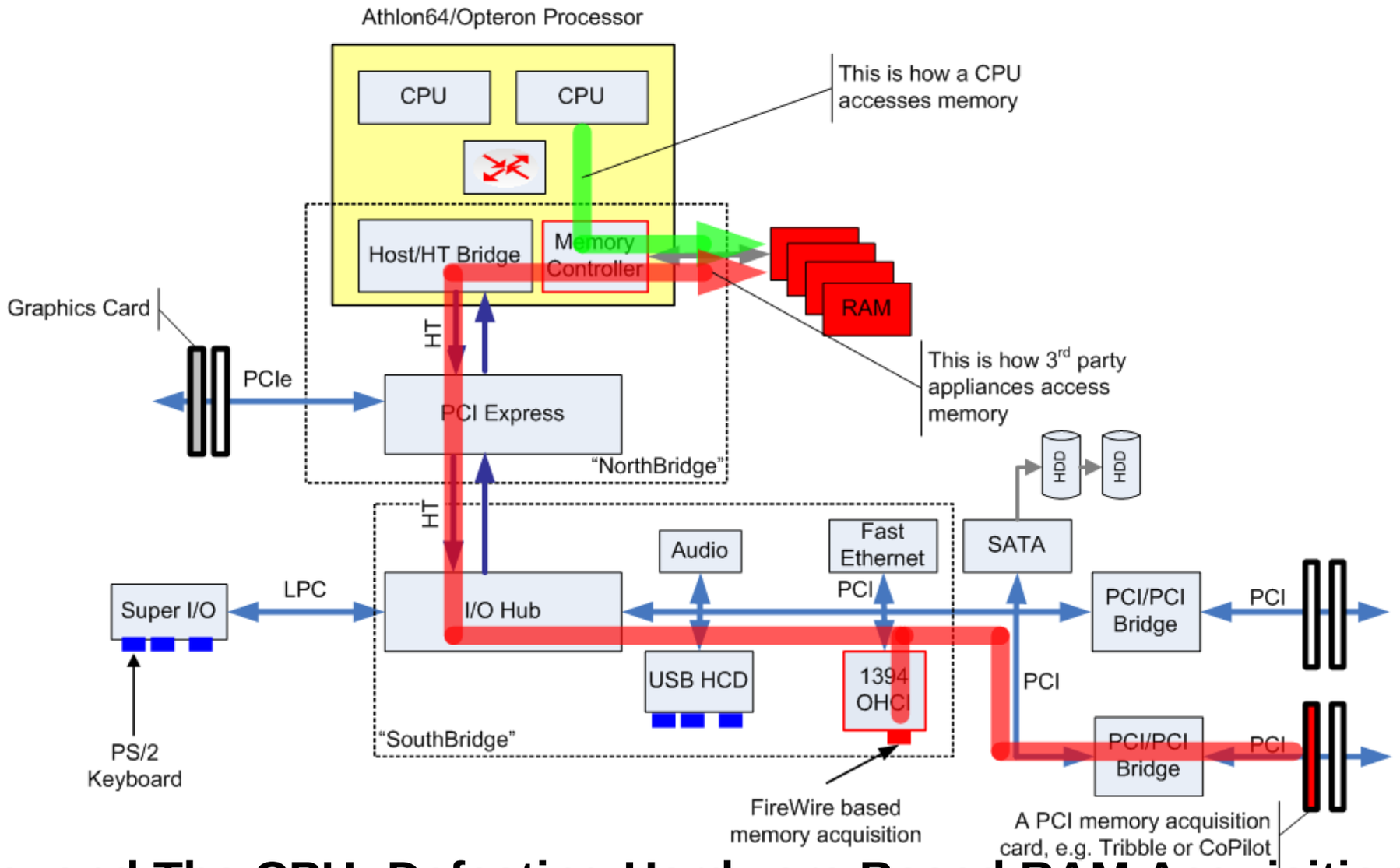


IOMMU

- The Input/output memory management unit (IOMMU) introduces virtual memory for external devices
- If properly configured, it can be used to prevent certain devices to access some range of memory
  - This is typically the case when a hypervisor is running
  - In this case, it is very hard to get a physical image of the entire memory

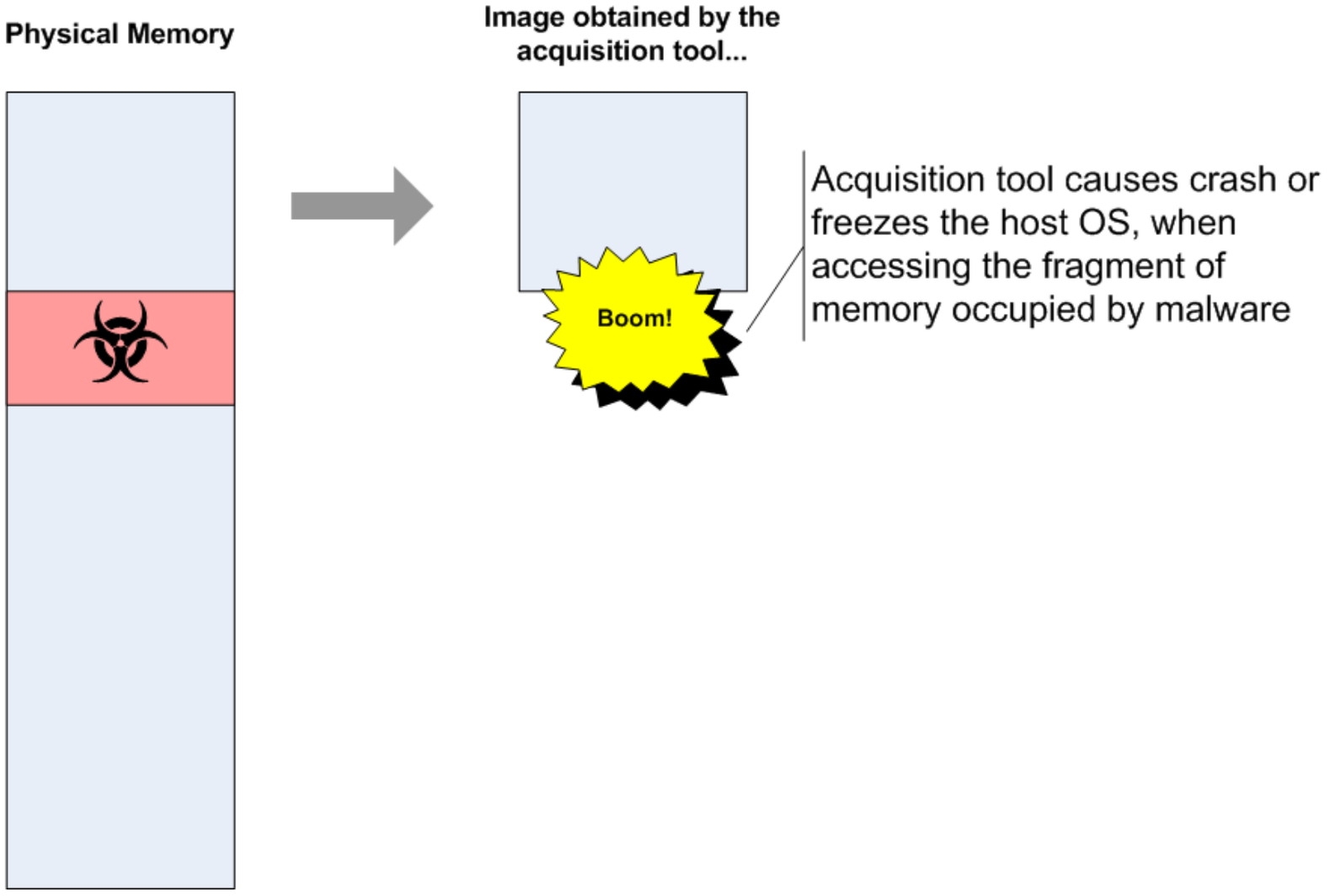


Accessing Physical Memory

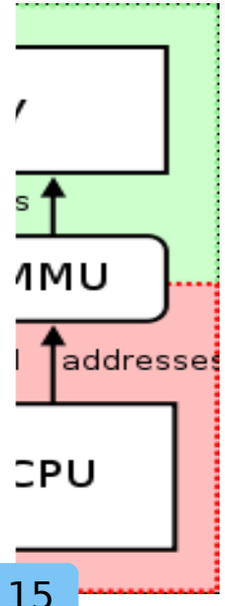


Beyond The CPU: Defeating Hardware Based RAM Acquisition (2007)

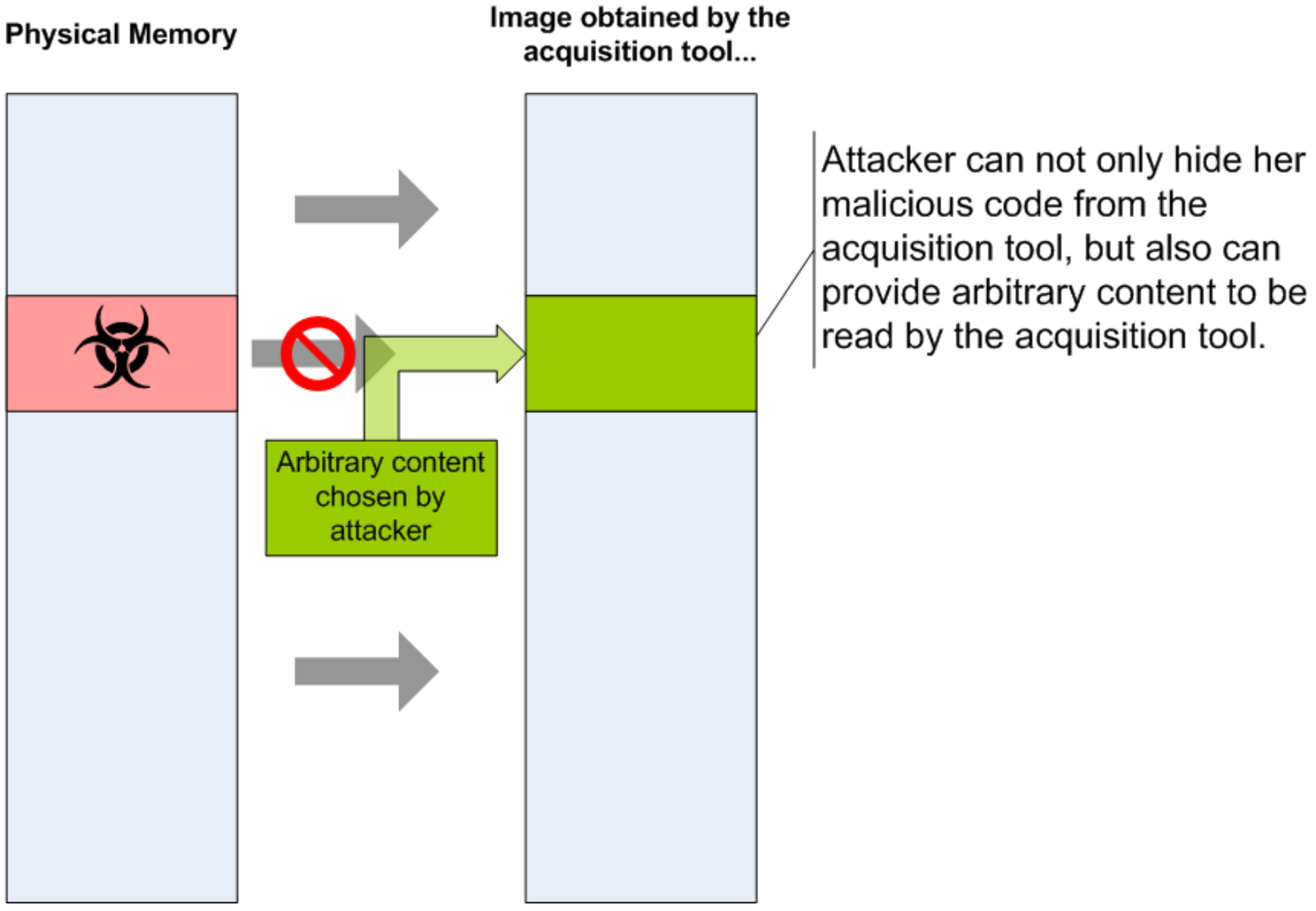
DoS Attack Illustration



- rtual
- ccess

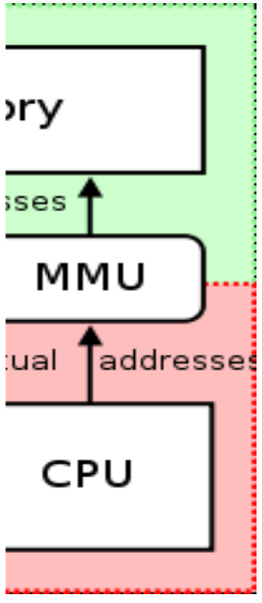


Full Replacing Attack Illustration



virtual

access



Software Acquisition

- Pseudo-device files
  - A file-like device that can be copied using `dd`
  - `/dev/mem` and `/dev/kmem` in Linux
    - Access to user- and kernel-memory
    - Not available on recent systems
  - `\\.\PhysicalMemory` in Windows 2000/2003/XP/Vista
    - Not accessible from user space since Windows 2003 SP1
  - Kernel-mode drivers exist to overcome the previous limitations (e.g., `win32dd` for Windows, `Lime`, `fmem`, `pmem.ko` for Linux)
  - Plenty of available tools: http://www.forensicswiki.org/wiki/Tools:Memory_Imaging

Memory Acquisition Internals

- Physical memory is not continuous
  - Don't expect a 4GB physical address space to span from `0x00000000` - `0xffffffff`
  - The physical address space is a virtual construct controlled by the Northbridge
  - Hardware peripherals map registers or parts of their integrated memory into the physical address space via Memory Mapped I/O
- The BIOS assign the RAM to various ranges in the physical address space
  - Any attempt to read the memory mapped to a device would probably crash the system
  - In Windows, most memory acquisition tools call `MmGetPhysicalMemoryRanges()` to get the physical memory ranges

Tools

`winpmem`

- Supports Windows XP SP2 to Windows10
- Outputs raw dumps or AFF4 dumps
  (an archive that can contain multiple streams and additional information)
- Live forensics using the Rekall framework
- Live memory modification of kernel data structure
- Examples:

```
> Winpmem_1.5.2.exe physmem.raw
> Winpmem_1.5.2.exe -d - | nc 10.0.0.10
```

- A version for Mac OS is also available (`OSXPmem`) and for Linux (`linpmem`)

Crash Dumps

- Windows can be configured to create a full memory dump in response to a *Blue Screen of Death* (BSoD)
  - Dumps are created in the swap area and then copied to regular files at the next boot
  - Memory dumps can be forced by pressing `CTRL + ScrollLock (x2)`
  - `notmyfault.exe` from SysInternals can cause a crash from software
- A full dump can be taken only if:
  - The swap file is at least 1 MB larger than the physical memory and it is located in the system volume (the one with the `\Windows` directory)
  - The corresponding key is set in the registry
    (not the default except for the Windows Server family)
  - If the RAM is > 2 GB, additional options must be set in `Boot.ini`
  - Instructions for a complete setup: http://support.microsoft.com/kb/969028

Crash Dumps

- Very accurate:
  the system is frozen while the dump is taken, allowing to take an atomic snapshot of the memory
- Impact on disk analysis:
  several GB are written to the disk, possibly overwriting other evidence
- Hard to deploy due to the several configuration options that need to be set in advance in the registry
  - Even worse, for some of them a system reboot is required to apply the changes

Hibernation Files

- When the OS is hybernated (suspended to disk) a copy of the RAM is stored in a file (`hiberfil.sys` in Windows)
- The `sandman` library (available for C and Python) can be used to read and write `hiberfil.sys` files, and convert them to raw memory dumps
  - http://www.msuiche.net/2008/02/26/sandman-10080226-is-out/
- If hybernation is supported, this is a good method to obtain a memory dump, with few limitations (e.g., it works in 64bit architecture with more than 4GB or RAM)
- The system can be hibernated by running:

```
> rundll32.exe powrprof.dll,SetSuspendState Hibernate
```

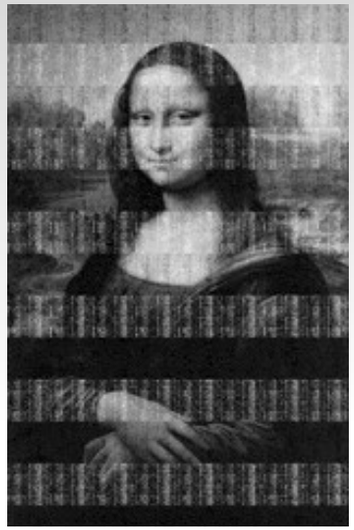
Cold Boot

- Main memory is normally stored in DRAM chips
  - Information is stored in a capacitor, whose charge needs to be refreshed every few milliseconds (or the charge would decay to the ground state)
  - If not refreshed, the content of DRAM is completely lost after several seconds (the actual time depends on the machine)
- Acquisition:
  - Cut the power of the target machine
  - Boot from network or from a USB drive
  - Copy the RAM – the process is relatively fast (~30sec per GB over the network to 4min over USB)
  - But what if the computer is not configured to boot from network or USB?

**Lest We Remember: Cold Boot Attacks On Encryption Keys*

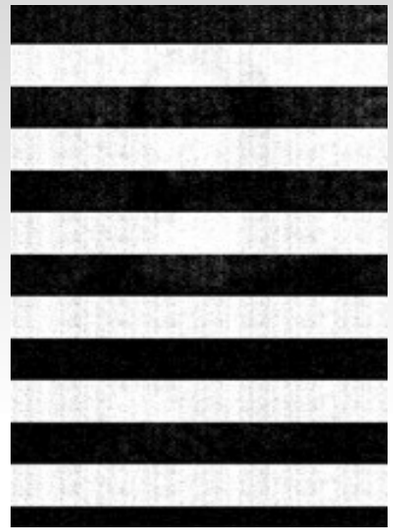
Memory Degradation at Room Temperature

After



5 seconds

30 seconds



60 seconds

5 minutes

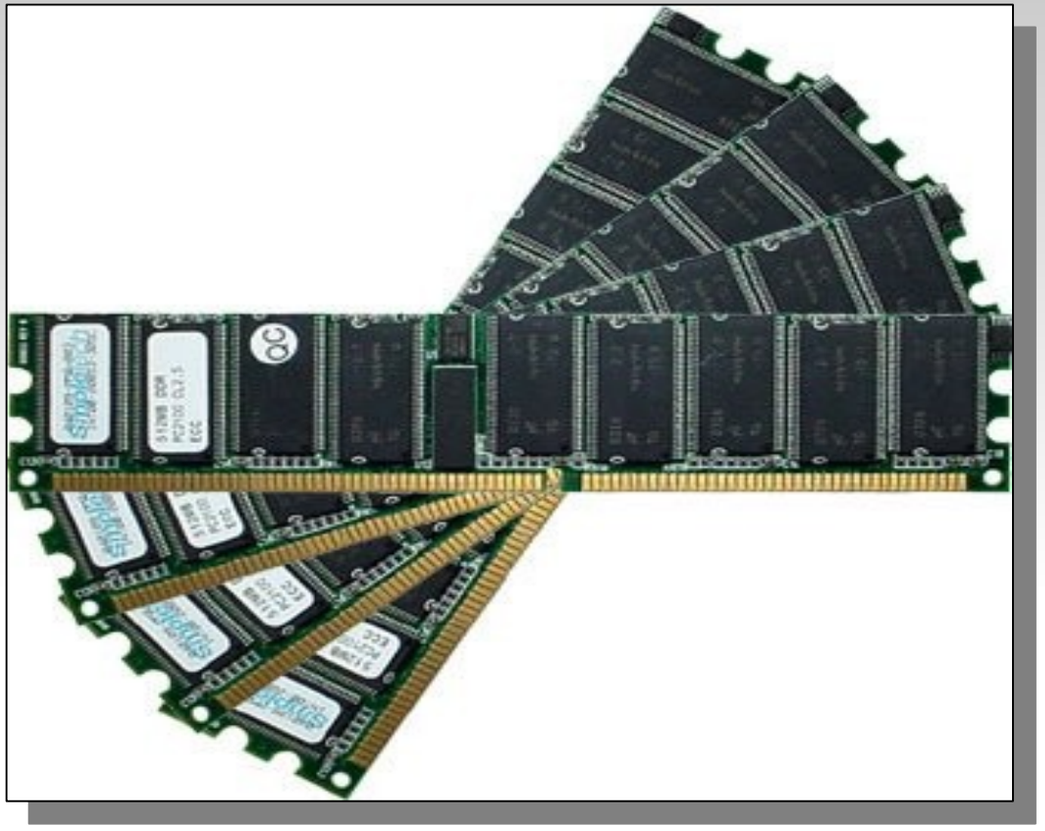
Mona Lisa on the Rocks

- A multi-purpose “canned air” duster spray canister, held upside-down, discharges very cold liquid refrigerant instead of gas
- It can be used as a fast and cheap refrigerant for memory chips :)
- 1% of bits decayed after 10 minutes

- Drop it in a liquid nitrogen can, and information is preserved unchanged for hours
  - 0.17% bits decayed in 60 minutes

On the Practicability of Cold Boot Attacks

- The attacker was originally designed for DDR1 and DDR2
- On DDR3, the memory controller scrambles the data before writing it to memory to reduce electromagnetic interference
  - Intel uses a Linear-feedback shift registers that can be reverted if a small plaintext is known
  - See “Lest we forget: Cold-boot attacks on scrambled DDR3 memory” (DFRWS EU 2016) for more information



Memory Analysis

Summary

- Strings and raw data
- Semantic Gap
  - Locating data structures
  - Address translation
- Volatility

Strings?

- A freshly booted machine roughly generates ~100MB of strings per GB of RAM*
  - ~580.000 strings of length 5
  - ~66.000 Unicode strings of length 5
- Starting notepad and IDA adds another ~7.000 new Unicode strings !!

- Much better to
  - focus on something in particular (IP addresses, email headers, …)
  - Whitelist strings of known binaries

**Post-Mortem RAM Forensics – Tim Vidas 2007*

Locating Structures

- Fixed offsets
  - Useful to find the kernel base image
- Data structure traversal (list walking, tree climbing, ..)
  - Extract allocated data structures
  - Requires knowledge of the kernel internals
- Linear scanning (~carving)
  - Search the memory for known patterns
  - It can detects de-allocated structures
  - Requires knowledge of the kernel internals
  - Fields validation to reduce false positives
    - Permitted values
    - Pointers target

Problems

- The modeling and extraction tools need a very precise definition of the kernel structures
- But kernel structures change quite rapidly and they are often unknown
  - In Windows, source code are not always available for documentation
  - In Linux, users can install different kernel, apply patches, or recompile the kernel from scratch
  - Automatic updates and service packs also update the kernel image

Interesting Structures

(from now on, we are talking about Windows)

- Each process is identified by an Executive Process Block (`EPROCESS`)
- All the `EPROCESS` are connected in a double linked list (`EPROCESS→ActiveProcessLinks→flink/blink`)
  - Processes are removed from the list when they exit
  - Rootkits often hide processes by taking them out of the linked list
- The `EPROCESS` structure contains a link to the Process Environment Block (PEB) located in the process address space

Process Creation

1. The executable file is opened and a section object is created
   (to be later mapped into the process address space)
2. The kernel sets up:
   - an `EProcess`, `KProcess`, and PEB structures
   - the process initial address space

   The executable section and `Ntdll.dll` are mapped to the new process address space
3. The initial thread is created
4. The Windows subsystem is notified of the new process and thread
5. The execution of the initial thread starts

Interesting Info

- Eprocess
  - Creation and Exit time
  - Process ID and parent Process ID (who started the process)
  - Pointer to the handle table
  - Virtual Address Space descriptors (VAD)
- PEB
  - Pointer to the image base address
    (where you can find the executable image)
  - Pointer to the process parameters structure
    (full path of binary, DLLs, and command line used to start the executable)
  - Pointer to the DLLs loaded by the process
    (three lists, ordered by loading time, initialization time, and memory address)
  - Heap size information
    (the pointer to the heap is located just after the PEB structure)

Interesting Info

- Eprocess
  - Creation and Exit time
  - Process ID and parent Process ID (who started the process)
  - Pointer to the handle table
  - Virtual Address Space descriptors (VAD)
- PEB
  - Pointer to the image base address
    (where you can find the executable image)
  - Pointer to the process parameters structure
    (full path of binary, DLLs, and command line used to start the executable)
  - Pointer to the DLLs loaded by the process
    (three lists, ordered by loading time, initialization time, and memory address)
  - Heap size information
    (the pointer to the heap is located just after the PEB structure)

Virtual Address Descriptors are structures used by the Memory Manager to keep track of all memory allocated in the system

Kernel Global Variables

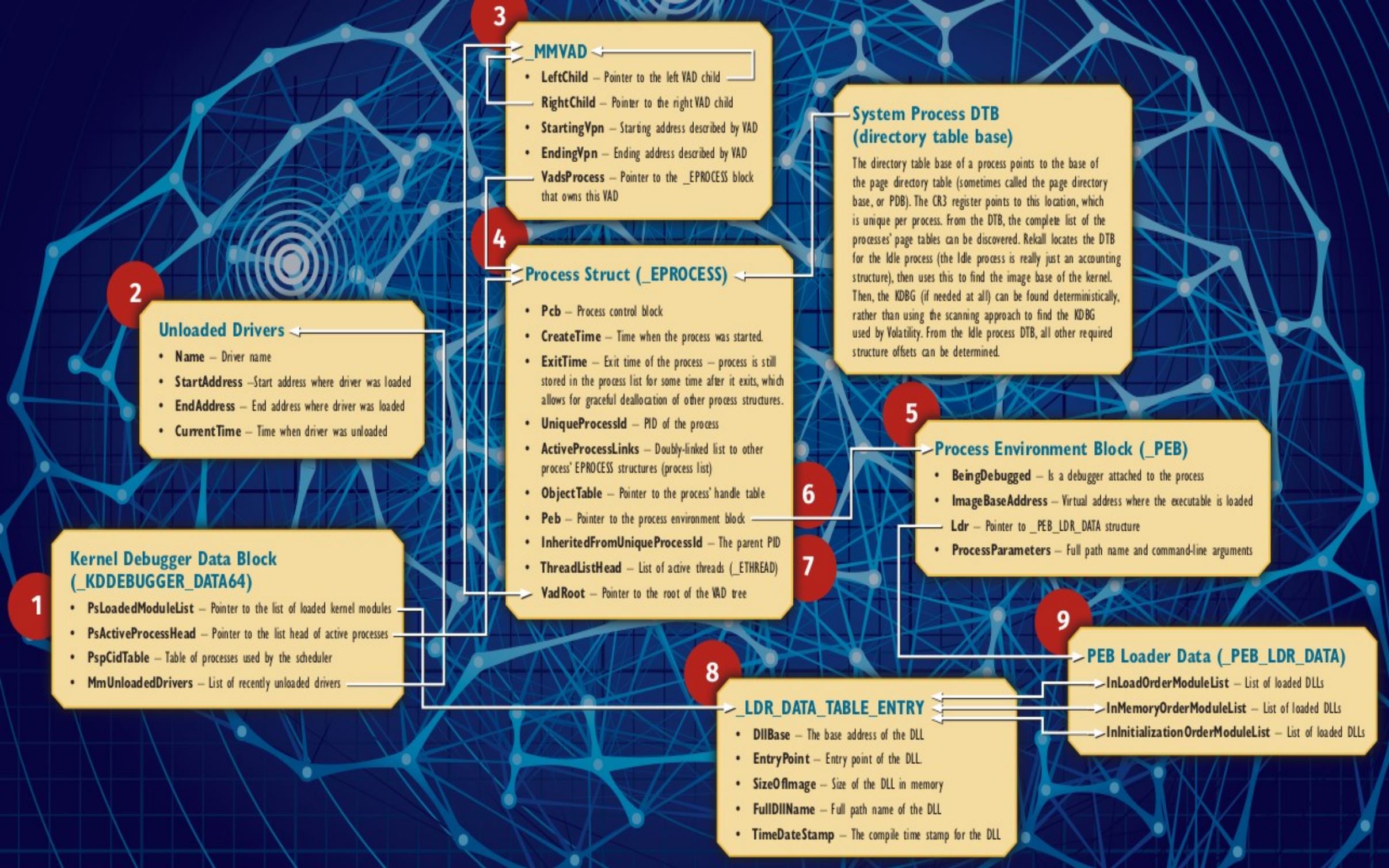
- A number of *hidden* kernel variables are extremely helpful to examine the state of the running system
  - `PsActiveProcessHead` points to the start of the kernel's list of `EPROCESS` structures
  - `PsLoadedModuleList` points to the list of currently loaded kernel modules
  - `HandleTableListHead` points to the head of list of handle tables (resources used by each process)
  - `MmPfnDatabase` is an array of structures describing each physical page in the system

Locating Kernel Variables

- The variables are always at a *fixed* location in memory
  - But unfortunately the location changes between Windows versions, patch levels, and even single hotfixes
- Windows keep a structure (`_KDDEBUGGER_DATA64`) for debugging purposes, that contains the memory address of dozens of global kernel variables
  - In Windows {XP, 2003, Vista} this structure can be found through a `KPCR` structure that is located at a fixed address in memory
  - In Windows 2000, the structure has to be located by scanning the memory
  - Windows 8 encodes the KDBG block making memory analysis more difficult

Bootstrapping the analysis process

- MS Windows
  - Volatility
    - Scans the memory to find the `KDBG` structure to locate the `PsActiveProcessHead`
  - Rekall
    - Scans the memory to find `RSDS` signature
    - Extracts `GUID` and `PDB` filename
    - Queries the Microsoft public symbols server
    - Extracts symbols from the `PDB` file
- Linux
  - Get all the addresses of the kernel data structures from the `system.map` file (a kernel symbol table)


3
_MMVAD
LeftChild – Pointer to the left VAD child
RightChild – Pointer to the right VAD child
StartingVpn – Starting address described by VAD
EndingVpn – Ending address described by VAD
VadsProcess – Pointer to the _EPROCESS block that owns this VAD
System Process DTB (directory table base)
The directory table base of a process points to the base of the page directory table (sometimes called the page directory base, or PDB). The CR3 register points to this location, which is unique per process. From the DTB, the complete list of the processes' page tables can be discovered. Rekall locates the DTB for the Idle process (the Idle process is really just an accounting structure), then uses this to find the image base of the kernel. Then, the KDBG (if needed at all) can be found deterministically, rather than using the scanning approach to find the KDBG used by Volatility. From the Idle process DTB, all other required structure offsets can be determined.
4
Process Struct (_EPROCESS)
Pcb – Process control block
CreateTime – Time when the process was started.
ExitTime – Exit time of the process – process is still stored in the process list for some time after it exits, which allows for graceful deallocation of other process structures.
UniqueProcessId – PID of the process
ActiveProcessLinks – Doubly-linked list to other process' EPROCESS structures (process list)
ObjectTable – Pointer to the process' handle table
Peb – Pointer to the process environment block
InheritedFromUniqueProcessId – The parent PID
ThreadListHead – List of active threads (_ETHREAD)
VadRoot – Pointer to the root of the VAD tree
2
Unloaded Drivers
Name – Driver name
StartAddress –Start address where driver was loaded
EndAddress – End address where driver was loaded
CurrentTime – Time when driver was unloaded
5
Process Environment Block (_PEB)
BeingDebugged – Is a debugger attached to the process
ImageBaseAddress – Virtual address where the executable is loaded
Ldr – Pointer to _PEB_LDR_DATA structure
ProcessParameters – Full path name and command-line arguments
6
7
1
Kernel Debugger Data Block (_KDDEBUGGER_DATA64)
PsLoadedModuleList – Pointer to the list of loaded kernel modules
PsActiveProcessHead – Pointer to the list head of active processes
PspCidTable – Table of processes used by the scheduler
MmUnloadedDrivers – List of recently unloaded drivers
9
PEB Loader Data (_PEB_LDR_DATA)
InLoadOrderModuleList – List of loaded DLLs
InMemoryOrderModuleList – List of loaded DLLs
InInitializationOrderModuleList – List of loaded DLLs
8
_LDR_DATA_TABLE_ENTRY
DllBase – The base address of the DLL
EntryPoint – Entry point of the DLL.
SizeOfImage – Size of the DLL in memory
FullDllName – Full path name of the DLL
TimeDateStamp – The compile time stamp for the DLL

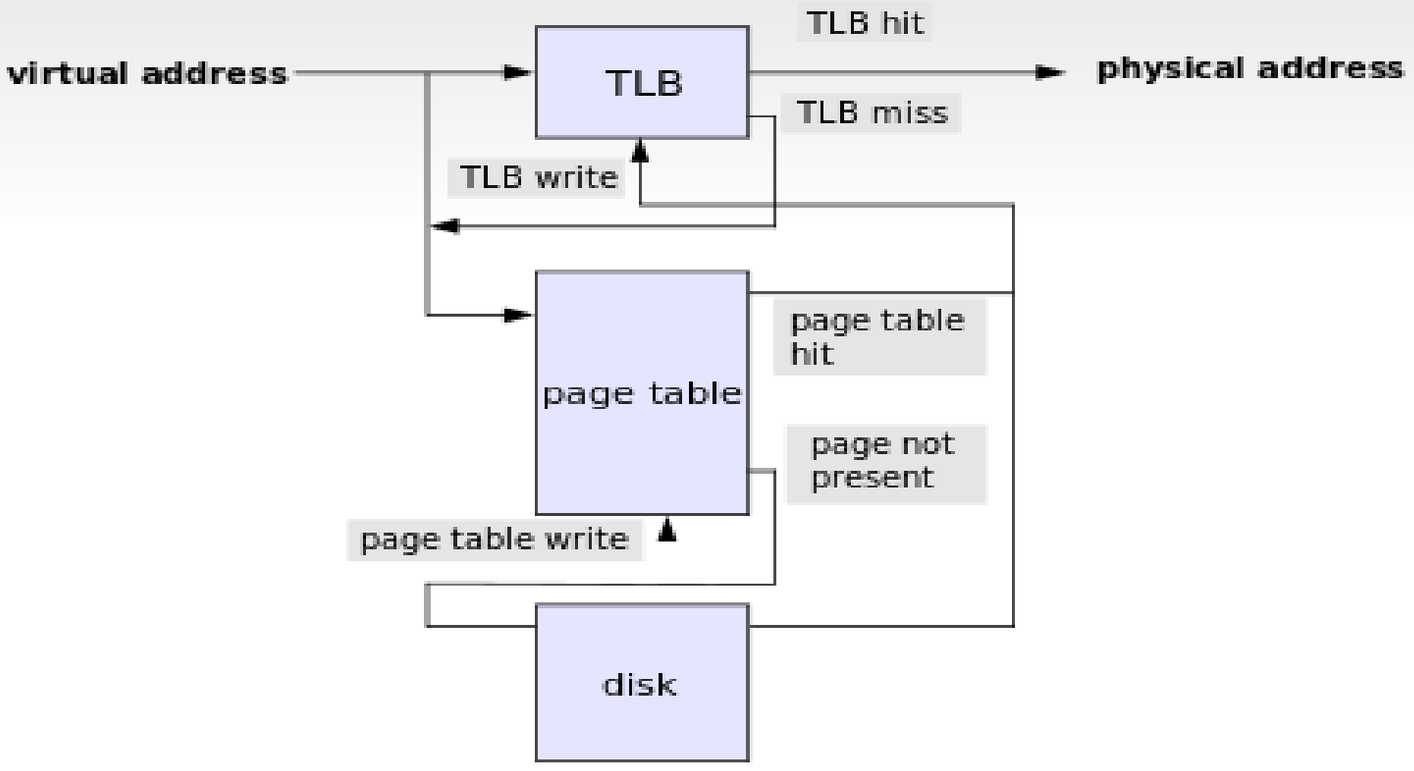
(Very) Short Introduction to Address Translation

- Memory analysis requires the ability to translate Virtual Addresses used by programs into the true memory locations in the memory image
- Memory is divided into pages (when in memory) or frames (when swapped to disk) of 4K each
- The OS presents to each program a large private virtual address space
  - Each time a program references a virtual address, the operating system translates that virtual address into a physical location and accesses the requested data
  - If necessary, the operating system loads data from the disk

** Using Every Part of the Buffalo in Windows Memory Analysis – Jesse D. Kornblum*

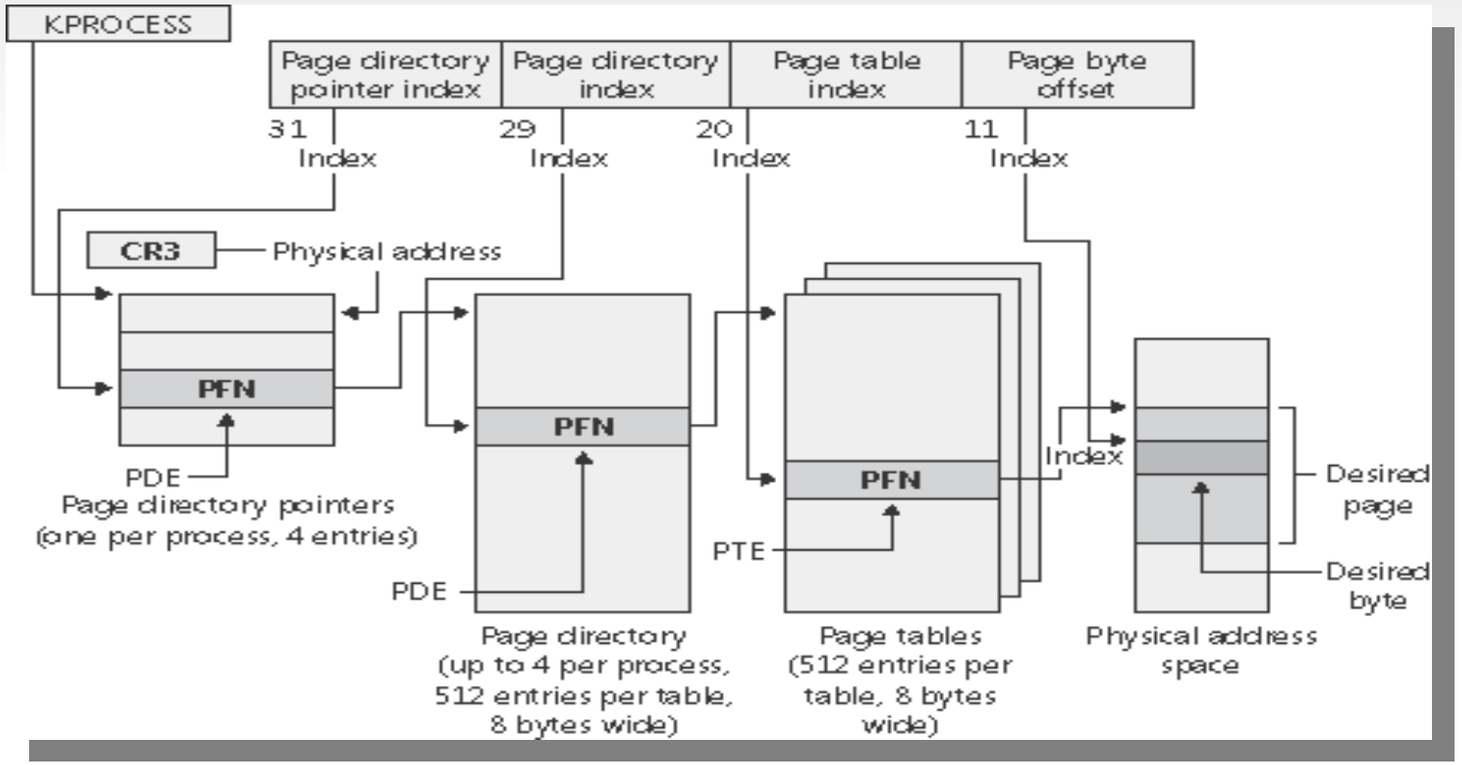
Address Translation

- The virtual memory system stores the mapping between logical and physical addresses inside Page Tables
  - Each mapping is called a Page Table Entry (PTE)



Address Translation in x86 32bit

- The actual address translation depends on the architecture (32bit or 64bit) and on certain options
  - Physical Address Extension (PAE)
  - Address Windowing Extensions (AWE)



Valid and Invalid Addresses

- The first 12 bits of both PDE and PTE entries contain a number of flags
  - Bit 1 – Validity flag (V)
  - Bit 10 – Prototype (P) bit
  - Bit 11 – Transition (T) bit
- If `(V==0)`, the address is not valid
  - If `(P==0 & T==0)` the entry points to a page file
  - If `(P==1)` the address belongs to a prototype page, shared between multiple processes
  - If `(P==0 & T==1)` the page is in memory but it is waiting to be written to the disk
- See also: http://rekall-forensic.blogspot.fr/2014/10/windows-virtual-address-translation-and.html

Summary

- ✔ Strings and raw data
- ✔ Semantic Gap
  - ✔ Locating data structures
  - ✔ Address translation
- ☐ Volatility

Volatility

- Open source memory analysis framework written in Python
- Version 2.6 (December 2016) supports
  - 32bit and 64bit Windows OSs {XP, Vista, 7, 2003, 2008, 8, 8.1, 2012}
  - 64-bit Windows 10 and Server 2016
  - Linux 32 and 64bit
  - MacOSX 10.5 to 10.12
  - Android
- Volatility supports raw dumps, crash dump, virtual machine snapshots, and hibernation files
  - The `imagecopy` plugins can convert any format to raw dd

Volatility Plugins

- Collection of tools implemented as plugins
- Plugins are just Python scripts and can be easily installed by copying them into the plugin directory
  - The current version contains 50 profiles and 265 plugins
  - A few plugins have been developed specifically to find signs of malware infections
    - http://malwarecookbook.googlecode.com/svn/trunk/malware.py
  - Additional (more research-oriented) plugins implemented by Brendan Dolan-Gavitt, a GeorgiaTech PhD student
    - http://www.cc.gatech.edu/~brendan/volatility/
  - Volatility plugins developed and maintened by the community:
    - https://github.com/volatilityfoundation/community
- `$ vol.py --info` → list the available plugins

Image Identification

```
$ vol.py imageinfo -f dump1.mem
Volatile Systems Volatility Framework 2.0
Determining profile based on KDBG search...


          Suggested Profile(s) : WinXPSP3x86, WinXPSP2x86
                     AS Layer1 : JKIA32PagedMemory (Kernel AS)
                     AS Layer2 : FileAddressSpace
                      PAE type : No PAE
                           DTB : 0x39000
                          KDBG : 0x8054ce60L
                          KPCR : 0xffdff000L
            KUSER_SHARED_DATA : 0xffdf0000L
           Image date and time : 2012-04-10 14:25:41
     Image local date and time : 2012-04-10 14:25:41
          Number of Processors : 1
                    Image Type : Service Pack 3
```

List of Processes

```
$ vol.py pslist --profile=WinXPSP3x86 -f dump1.mem
 Offset(V)  Name            PID    PPID   Thds   Hnds   Time
---------- ------------ ------ ------ ------ ------ -------------------
0x823c89c8 System            4      0     54    644  1970-01-01 00:00:00
0x82104020 smss.exe        520      4      3     19  2012-04-10 23:13:25
0x820a6da0 csrss.exe       584    520     10    361  2012-04-10 23:13:26
...
0x822e7358 notepad.exe     980   1612      1     27  2012-04-10 14:28:44
0x822cdda0 idaq.exe        176   1612      5    287  2012-04-10 14:29:53
```

- `pslist` walks the `EProcess` list
- `pstree` does the same but it prints the process tree
- `psscan` performs a linear scan to find data that looks like an `EProcess` structure (less reliable, but it can find hidden and terminated processes)
- `-P` switch prints the physical (instead of virtual) addresses

DLLs

- List all the DLLs loaded by a process:

```
$ vol.py –profile=… -f dump1.mem dlllist –pid=492
```

- Dump a DLL:

```
$ vol.py –profile=… -f dump1.mem dlldump –pid=492
```

- DLL inside an hidden process ??

```
$ vol.py –profile=… -f dump1.mem dlldump -o 0x3e3f64e8
                                          --base=0x00680000
```

DLLs

- List all the DLLs loaded by a process:

```
$ vol.py –profile=… -f dump1.mem dlllist –pid=492
```

- Dump a DLL:

```
$ vol.py –profile=… -f dump1.mem dlldump –pid=492
```

- DLL inside an hidden process ??

```
$ vol.py –profile=… -f dump1.mem dlldump -o 0x3e3f64e8
                                          --base=0x00680000
```

can also be set in environment variables or configuration files

And more..

- Open handles:

```
$ vol.py –profile=… -f dump1.mem handles
```

- Dump process memory:

```
$ vol.py –profile=… -f dump1.mem memdump -p pid
```

- List open network connections:

```
$ vol.py –profile=… -f dump1.mem connections
$ vol.py –profile=… -f dump1.mem netscan
```

- List kernel modules:

```
$ vol.py –profile=… -f dump1.mem modules
```

Processes Memory

Command	Description	OS Support
memmap	Print the virtual addresses, physical addresses, and size of each page accessible to a process	All
memdump	Dump the addressable memory for a process (outputs 1 file per process)	All
procmemdump	Extract a process's executable, preserving slack space	All
procexedump	Extract a process's executable, do not preserve slack space	All
vadwalk	Walk the VAD tree and print basic information	All
vadtree	Walk the VAD tree and display in tree format	All
vadinfo	Walk the VAD tree and print extended information	All
vaddump	Dumps out the VAD sections (outputs multiple files per process)	All

Kernel Memory and Objects

Command	Description	OS Support
modules	Print loaded kernel drivers by walking the PsLoadedModuleList linked list	All
modscan	Scan physical memory for LDR_DATA_TABLE_ENTRY objects. Can locate unloaded and unlinked kernel drivers.	All
moddump	Extract a kernel driver to disk (by base address or regular expression)	All
ssdt	Print the Native and GDI System Service Descriptor Tables	All
driverscan	Scan physical memory for DRIVER_OBJECT objects	All
filescan	Scan physical memory for FILE_OBJECT objects	All

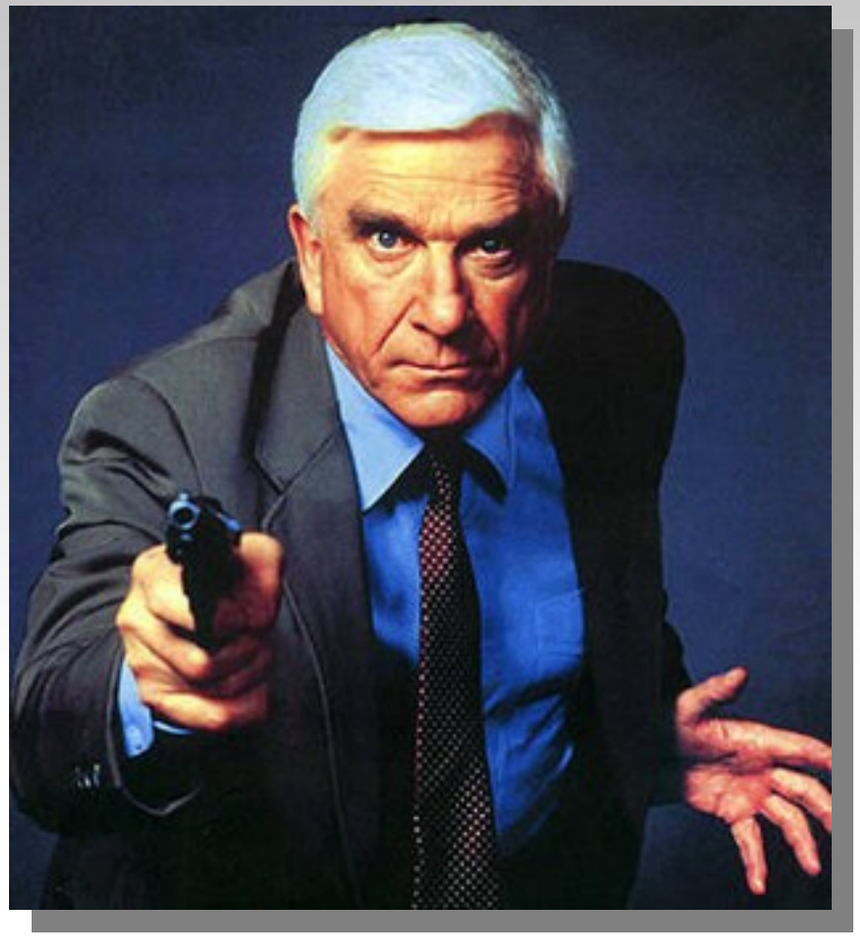
https://github.com/volatilityfoundation/volatility/wiki/Command-Reference

Rekall

- A fork of Volatility from the Google forensics team
  - Very active team that added many new features
  - Faster and improved architecture
  - Includes memory acquisition tools for Linux, Windows, and OS X
  - Native support for live forensics
  - Python interactive shell with support for sessions
    - Temporary objects and results are stored in the current session
    - Session can be saved and re-loaded later
  - Native support for Vms and Hypervisor forensics
    (porting of the Actaeon tool developed @ Eurecom)

Actaeon

- Plugin for Volatility to perform memory forensics in presence of one or more hypervisors
- Achieve three important goals:
  - locate any Hypervisor that uses the Intel VT-x technology
  - detect and analyze nested virtualization and show the relationships among different hypervisors running on the same machine
  - provide a transparent mechanism to recognize and support the address space of the virtual machines
- Developed at Eurecom
  (it started as a student project in the Forensics class!)
  - http://s3.eurecom.fr/tools/actaeon/

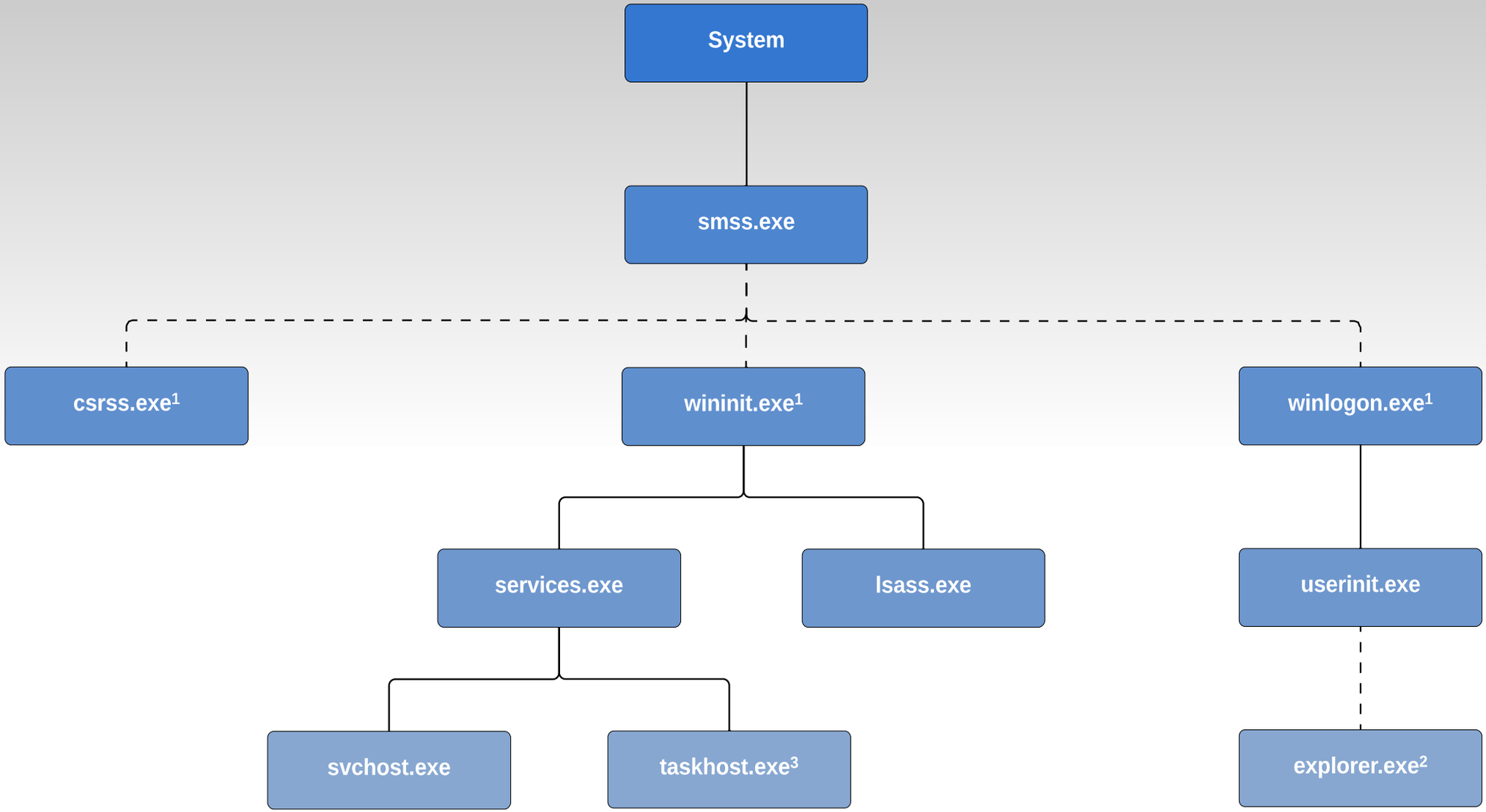


Catching the bad Guys

Memory Analysis

- The analysis often starts by listing and investigating the processes that were running in the system
  - Open files, loaded DLLs, or network sockets can help identifying suspicious cases
  - The starting point for the analysis may come from another source (e.g., a network sensor detected a suspicious connection)
- The analysis also includes the inspection of the kernel, to locate malicious kernel modules
- The analysis may end when you locate a known malicious file, or dump an unknown suspicious file that require some further binary analysis

Windows Process Genealogy



https://www.13cubed.com/downloads/windows_process_genealogy.pdf

csrss.exe

on.exe[1]

nit.exe

rer.exe[2]

Windows Process Genealogy

lsass.exe

Image Path: `%SystemRoot%\System32\lsass.exe`

Parent Process: `wininit.exe`

Number of Instances: One

User Account: Local System

Start Time: Within seconds of boot time

Description: The Local Security Authentication Subsystem Service process is responsible for authenticating users by calling an appropriate authentication package specified in **`HKLM\SYSTEM\CurrentControlSet\Control\Lsa`**. Typically, this will be Kerberos for domain accounts or MSV1_0 for local accounts. In addition to authenticating users, **`lsass.exe`** is also responsible for implementing the local security policy (such as password policies and audit policies) and for writing events to the security event log. Only one instance of this process should occur and it should not have child processes.

Detecting Malicious Processes

Hidden through DKOM (Direct Kernel Object Manipulation),
by removing the process from the `EProcess` linked list

- Compare the output of the `plist` and `pscan` plugins
- `Psxview` outputs the list of process extracted in six different ways

Detecting Malicious Processes

Disguised by renaming the process to match a system or innocuous one

- Use `dlllist` to see the full path of the executable
- Use `ldrmodules` to detect unlinked DLLs that are not listed in the PEB lists

Investigating Process Memory

Use the `vadinfo` plugin to analyze the metadata associated to various regions of process memory

```
VAD node @821b2790 Start 00400000 End 0066efff Tag Vad
Flags: ImageMap
Commit Charge: 13 Protection: 7
ControlArea @8223f818 Segment e16e3008
Dereference list: Flink 00000000, Blink 00000000
NumberOfSectionReferences: 1 NumberOfPfnReferences: 362
NumberOfMappedViews:       1 NumberOfUserReferences: 2
WaitingForDeletion Event:  00000000
Flags: Accessed, File, HadUserReference, Image
FileObject @8223f83c FileBuffer @ e15f10f8
Name: \Program Files\IDA\idaq.exe
First prototype PTE: e16e3040 Last contiguous PTE: fffffffc
Flags2: Inherit
File offset: 00000000
```

Investigating Process Memory

Use the `vadinfo` plugin to analyze the metadata associated to various regions of process memory

```
VAD node @821b2790 Start 00400000 End 0066efff Tag Vad
Flags: ImageMap
Commit Charge: 13 Protection: 7
ControlArea @8223f818 Segment e16e3008
Dereference list: Flink 00000000, Blink 00000000
NumberOfSectionReferences: 1 NumberOfPfnReferences: 362
NumberOfMappedViews:       1 NumberOfUserReferen
WaitingForDeletion Event:  00000000
Flags: Accessed, File, HadUserReference, Image
FileObject @8223f83c FileBuffer @ e15f10f8
Name: \Program Files\IDA\idaq.exe
First prototype PTE: e16e3040 Last contiguous PTE
Flags2: Inherit
File offset: 00000000
```

```
0 – No Access
1 – ReadOnly
2 – Execute
3 – Exec-Read
4 – Read-Write
5 – Write-Copy
6 – Exec-Read-Write
7 – Exec-Write-Copy
```

Detecting Injected DLLs

Injecting a DLL inside another process is a very common way for malware to hide their presence by not showing up in the process list

- Examine the VAD for areas associated to DLLs
  - Regularly loaded DLLs are associated to entries of type `Vad`, or `VadL` (long)
  - Pages allocated with `VirtualAllocEx` or `WriteProcessMemory` are of type `VadS` (short)
  - Even more suspicious if the page permissions are R-W-E
- The `malfind` plugin is automatically searching for these cases

API Hooks

Hooking other processes API is another common malware functionality.
It can be done in two main ways:

- by modifying their *Import Address Table* (IAT) of the target process
- with inline hooking, by replacing few instructions in the function prologues

The `apihooks` plugin can be used to detect both IAT
and inline hooking, showing where each function is redirected

Extracting Malicious Code

Hollow Process: a legitimate process is loaded on the system to serve as a container for the malware.
The legitimate code is completely replaced by the malicious code

---

- Hollow processes are not easy to detect
- Volatility can be used to extract the executable image from a process, by using
  - `procexedump` (removes the slack space)
  - `procmemdump` (keeps the memory as it is)
- The result is not an exact copy of the binary on disk, but it is enough to perform some BINARY ANALYSIS

Demo time!