Finding Digital Evidence In Physical Memory

Mariusz Burdach

Overview

- Introduction
- Anti-forensics
- Acquisition methods
- Windows memory analysis
- Linux memory analysis
- Detecting hidden data on a live system
- Q & A

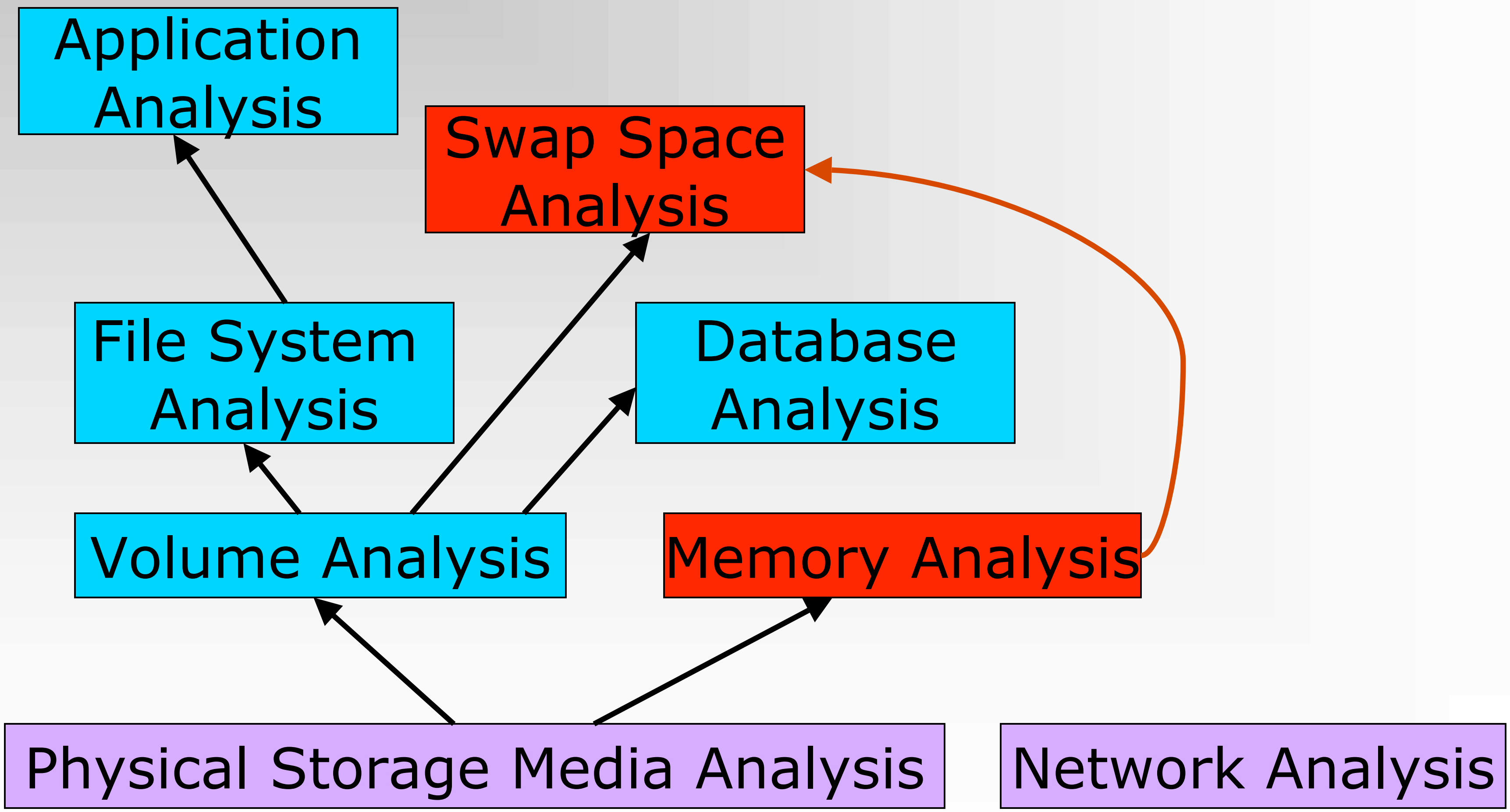
Past, Present & Future

- Forensic Analysis = File System Forensic Analysis
  - Well-developed procedures for seizing digital evidence from hard disk (i.e. Imaging a hard disk)
  - Quite difficult to tamper evidence during collecting data
  - Well-known methods of analysis

Past, Present & Future

- Some evidence is temporary stored in swap space
- Some evidence resides only in storages (i.e. volatile memory)
- Anti-forensics
  - Data contraception
  - Data hiding
  - Data destruction

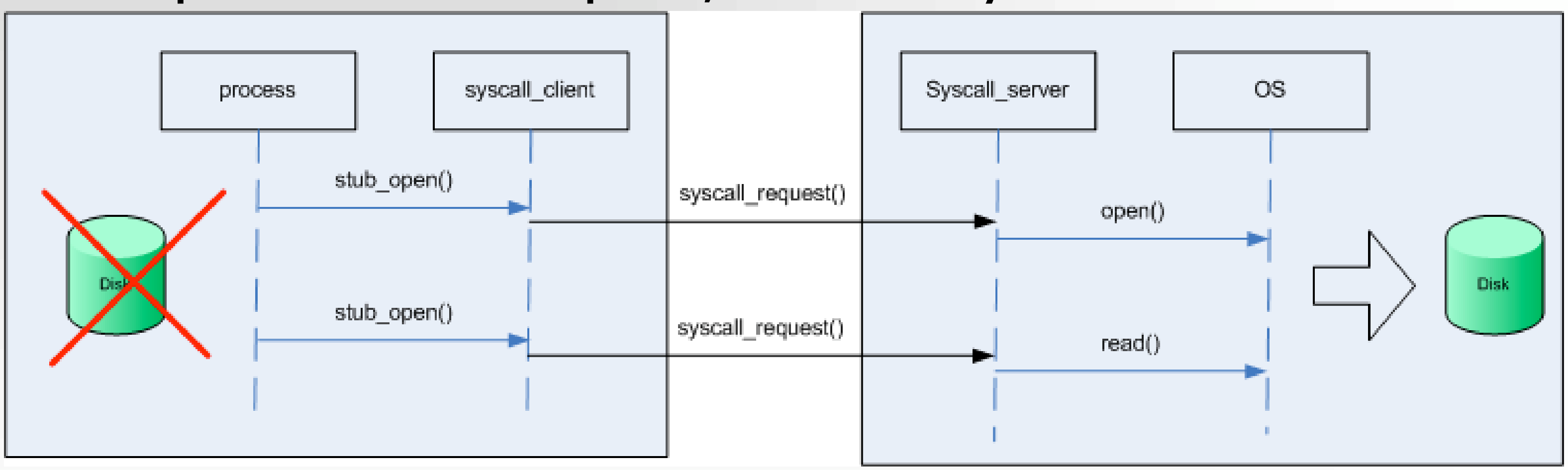
Analysis Types



Source: „File System Forensic Analysis", Brian Carrier

Anti-forensics

- Syscall proxying - it transparently „proxies" a process' system calls to a remote server:
  - Examples: CORE Impact, Immunity CANVAS



- In-Memory Library Injection – a library is loaded into memory without any disk activity:
  - Metasploit's Meterpreter (e.g. SAM Juicer)

Anti-forensics

- Anti-forensic projects focused on data contraception:
  - „Remote Execution of binary without creating a file on disk” by grugq (Phrack #62)
  - „Advanced Antiforensics : SELF” by Pluf & Ripe (Phrack #63)

Anti-forensics

- Advanced rootkits
  - Evidence gathering or incident response tools can be easily cheated
  - Examples: Hacker Defender/Antidetection, FU/Shadow Walker
- In memory worms/rootkits
  - Their codes exist only in a volatile memory and they are installed covertly via an exploit
  - Example: Witty worm (no file payload)

Past, Present & Future

- If it is possible – a physical memory from a suspicious computer has to be collected
- The operating system swaps out constantly some data from a physical memory to hard disk
- During forensic analysis of file systems we could correlate data from swap space with data which is resident in a main memory

How to acquire volatile data?

- All data in a main memory is volatile – it refers to data on a live system. A volatile memory loses its contents when a system is shut down or rebooted
- It is impossible to verify an integrity of data
- Acquisition is usually performed in a timely manner (Order of Volatility - RFC 3227)
- Physical backup instead of logical backup
- Volatile memory acquisition procedures can be:
  – Software-based
  – Hardware-based

Software-based methods

- Software-based memory acquisitions:
  - A trusted toolkit has to be used to collect volatile data
  - Every action performed on a system, whether initiated by a person or by the OS itself, will alter the content of memory:
    - The tool will cause known data to be written to the source
    - The tool can overwrite evidence
  - It is highly possible to cheat results collected in this way

Hardware-based methods

- Hardware-based memory acquisitions:
  - We can access memory without relying on the operating system, suspending the CPU and using DMA (Direct Memory Access) to copy contents of physical memory (e.g. TRIBBLE – PoC Device)
    - Related work (Copilot Kernel Integrity Monitor, EBSA-285)
  - The FIREWIRE/IEEE 1394 specification allows clients’ devices for a direct access to a host memory, bypassing the operating system (128 MB = 15 seconds)
    - Example: Several demos are available at http://blogs.23.nu/RedTeam/stories/5201/ by RedTeam

Physical Memory Devices

- **\\.\PhysicalMemory** - device object in Microsoft Windows 2000/2003/XP
- **/dev/mem** – device in many Unix/Linux systems
- **/proc/kcore** – some pseudo-filesystems provides access to a physical memory through /proc
- Software-based acquisition procedure
  - dd.exe if=\\.\PhysicalMemory of=\\<*remote_share*>\memorydump.img
- DD for Windows - Forensic Acquisition Utilities is available at http://users.erols.com/gmgarner/forensics/
- DD for Linux by default included in each distribution (part of GNU File Utilities)

Projects

- Web page: http://forensic.seccure.net
- Analysis of Windows memory images
  - **WMFT** - Windows Memory Forensics Toolkit
  - Written in C#
  - .NET 2.0 Framework
- Analysis of Linux memory images
  - gdb tool is enough to analyze a memory image, but we can simplify some tasks by using the **IDETECT** toolkit
- These tools could be used on a live system as an integral part of incident response toolkit

DFRWS Challenge 2005

- Digital Forensic Research WorkShop
- The Memory Analysis Challenge
- Results: 2 new tools
  - Memparser reconstructs a process list and extracts information from a process memory (Chris Betz)
  - Kntlist interprets structures of memory (George M. Garner Jr. and Robert Jan Mora)

Related work

- Memparser by Chris Betz
  - Enumerates processes (PsActiveProcessList)
  - Dumps process memory to disk
  - Dumps process strings to disk
  - Displays Process Environment Information
  - Displays all DLLs loaded by process

Related work

- Kntlist by George M. Garner Jr. and Robert Jan Mora
  - Copies, compresses, creates checksums & sends a physical memory to a remote location
  - Enumerates processes (PsActiveProcessList)
  - Enumerates handle table
  - Enumerates driver objects (PsLoadedModuleList)
  - Enumerates network information such as interface list, arp list, address object and TCB table
  - References are examined to find hidden data
    - Object table, its members and objects inside object directory point to processes and threads
    - Enumerates contents of IDT, GDT and SST to identify loaded modules

Preparation

- Useful files (acquired from a file system):
  - Kernel image file
  - Drivers/modules
  - Configuration files (i.e. SAM file, boot.ini)
- These files must be trusted
  - File Hash Databases can be used to compare hash sums
- Map of Symbols
  - System.map file
  - Some symbols are exported by core operating system files

Terminology

- Data – content of objects (data block | page frame)

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00010000	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZ□
00010010	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00	¸ @
00010020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00010030	00	00	00	00	00	00	00	00	00	00	00	00	D8	00	00	00	Ř
00010040	0E	1F	BA	0E	00	B4	09	CD	21	B8	01	4C	CD	21	54	68	ş ´ Í!¸ LÍ!Th
00010050	69	73	20	70	72	6F	67	72	61	6D	20	63	61	6E	6E	6F	is program canno
00010060	74	20	62	65	20	72	75	6E	20	69	6E	20	44	4F	53	20	t be run in DOS

- Metadata – provides details about any given object (i.e. internal data structures)

```
kd> dt _EPROCESS 8932cda0
        +0x000 Pcb                    : _KPROCESS
        +0x06c ProcessLock            : _EX_PUSH_LOCK
        +0x070 CreateTime             : _LARGE_INTEGER 0x1c60ac5`b38bb370
        +0x078 ExitTime               : _LARGE_INTEGER 0x0
        +0x080 RundownProtect         : _EX_RUNDOWN_REF
        +0x084 UniqueProcessId        : 0x00000b00
        +0x088 ActiveProcessLinks     : _LIST_ENTRY [ 0x89267e28 - 0x89a7bc20 ]
        ...
```

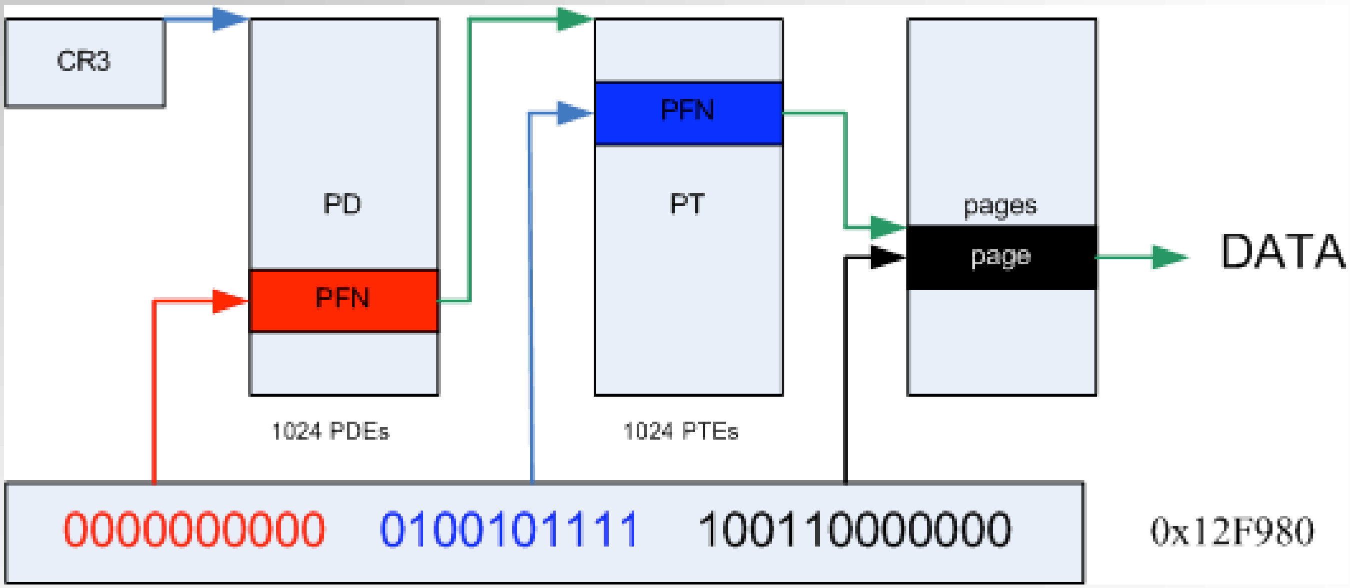
Methods of analysis

- String searches – extracting strings from images
  - ASCII & UNICODE
- Signature matching – identifying memory mapped objects by using fingerprints (e.g. file headers, .text sections)
- Interpreting internal kernel structures
  - This is a very easy task on systems with the source code
  - Analysis against Microsoft Windows systems is more challenging
    - For example: Windows NT family
    - Symbols from MS web site + Livekd from Sysinternals are to find some addresses (we have to be sure that a version of operating systems are the same)
- Enumerating & correlating all page frames

Windows memory analysis

- Information about the analyzed memory dump
  - The size of a page = 0x1000 bytes
  - Physical Address Extension (PAE)
  - Architecture 32-bit/64-bit/IA-64
- Memory layout
  - Virtual Address Space/Physical Address Space
  - User/Kernel land (2GB/2GB by default)
    - Kernel offset at 0x80000000
  - The PFN Database at 0x80c00000
  - The PTE Base at 0xC0000000
  - Page directory – each process has only one PD
- Knowledge about internal structures is required

Virtual To Physical Address Translation



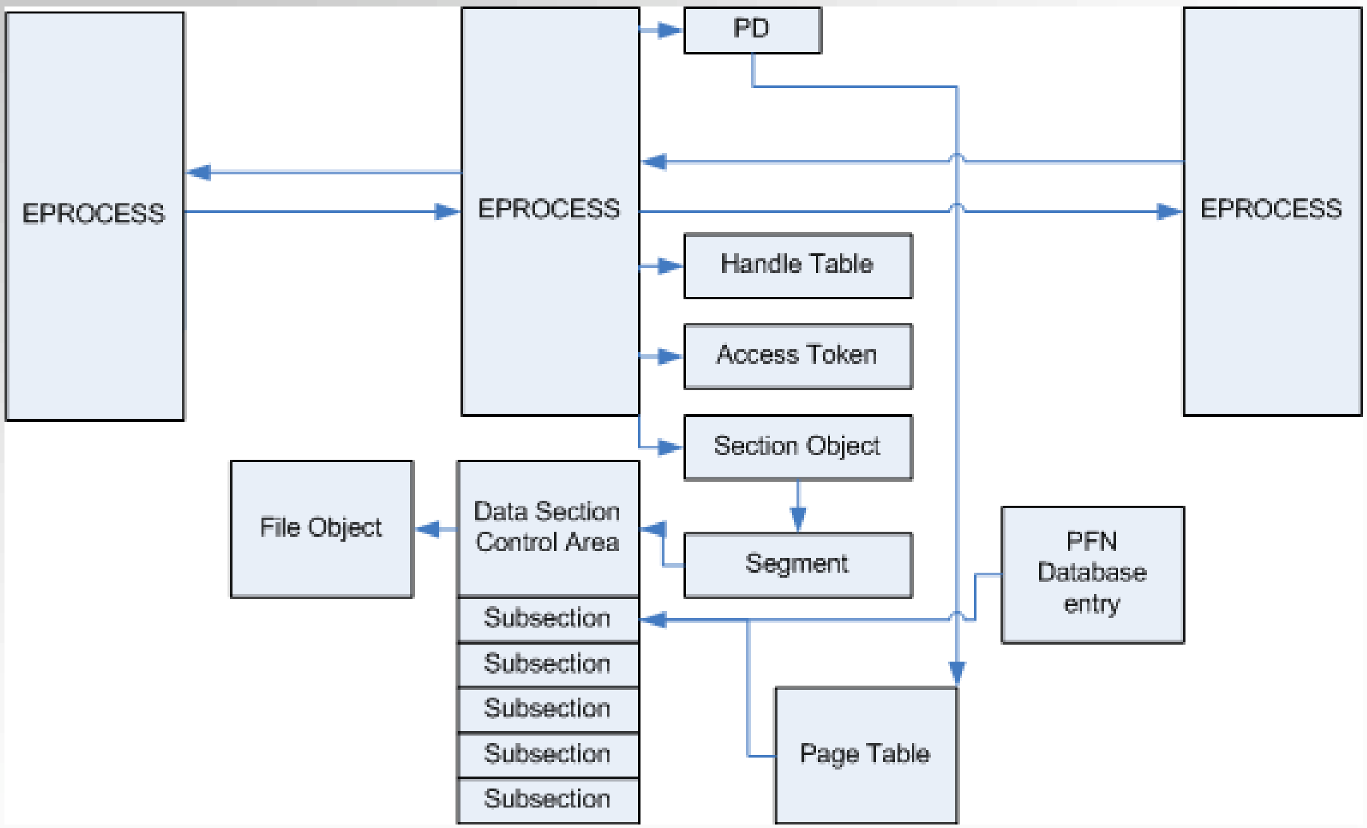
PTE address = PTE_BASE + (page directory index) * PAGE_SIZE

+ (page table index) * PTE size

Important kernel structures

- EPROCESS (executive process) block
- KPROCESS (kernel process) block
- ETHREAD (executive thread) block
- ACCESS_TOKEN & SIDs
- PEB (process environment) block
- VAD (virtual address descriptor)
- Handle table
- PFN (Page Frame Number Entries) & PFN Database
- Page frames
  - PTE_BASE, PAGE_DIRECTORY & PAGE_TABLES

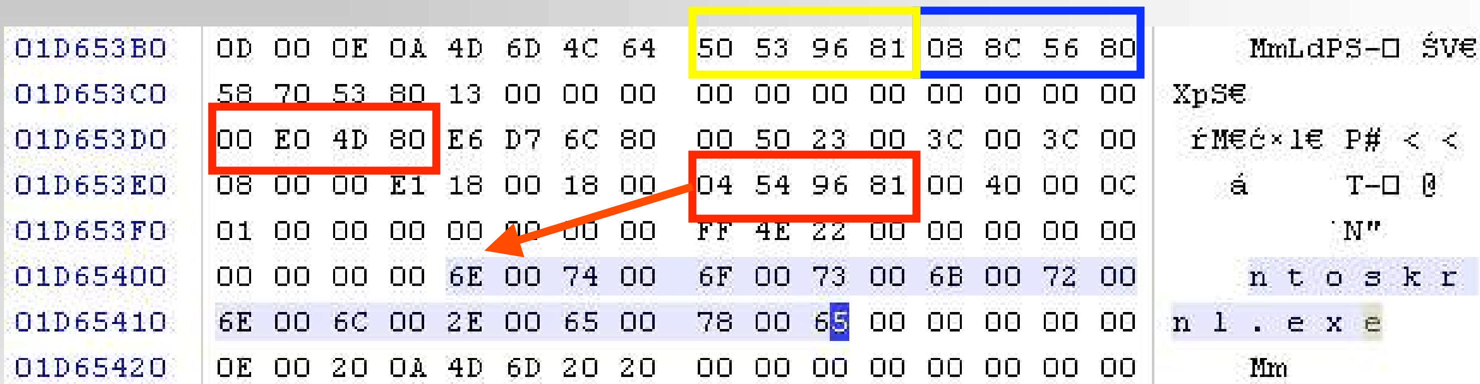
Relations between structures



Identifying core addresses

- Finding physical address (PA) of memory mapped kernel
  - Kernel image file: ntoskrnl.exe
  - Portable Executable (PE) file format
  - Base Address (typically 0x00400000)
  - Kernel offset = 0x80000000 (VA)
  - ntoskrnl.exe – first module on PsLoadedModuleList
- MODULE_ENTRY object
  - 0x0 -> LIST_ENTRY module_list_entry;
  - 0x18 -> DWORD driver_start;
  - 0x30 -> DWORD UNICODE_STRING driver_name;
- Extracting the „ntoskrnl.exe” string from the image
- Base Address and Kernel Image Address are used to calculate various addresses

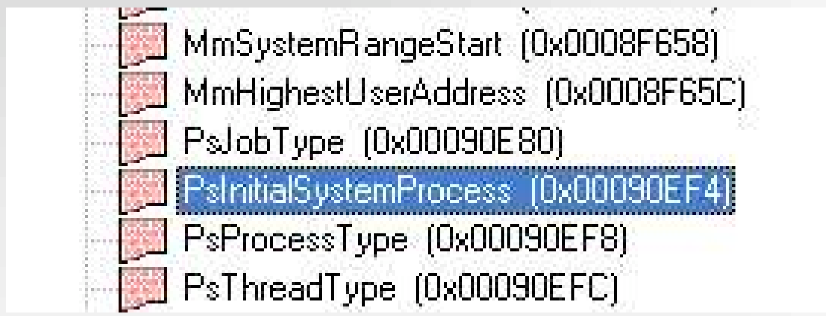
Identifying core addresses



- VA (0x81965404) = PA (0x1D65404)
- driver_start (VA) = 0x804DE000
- Kernel image is loaded at (PA) 0x004DE000

Enumerating processes

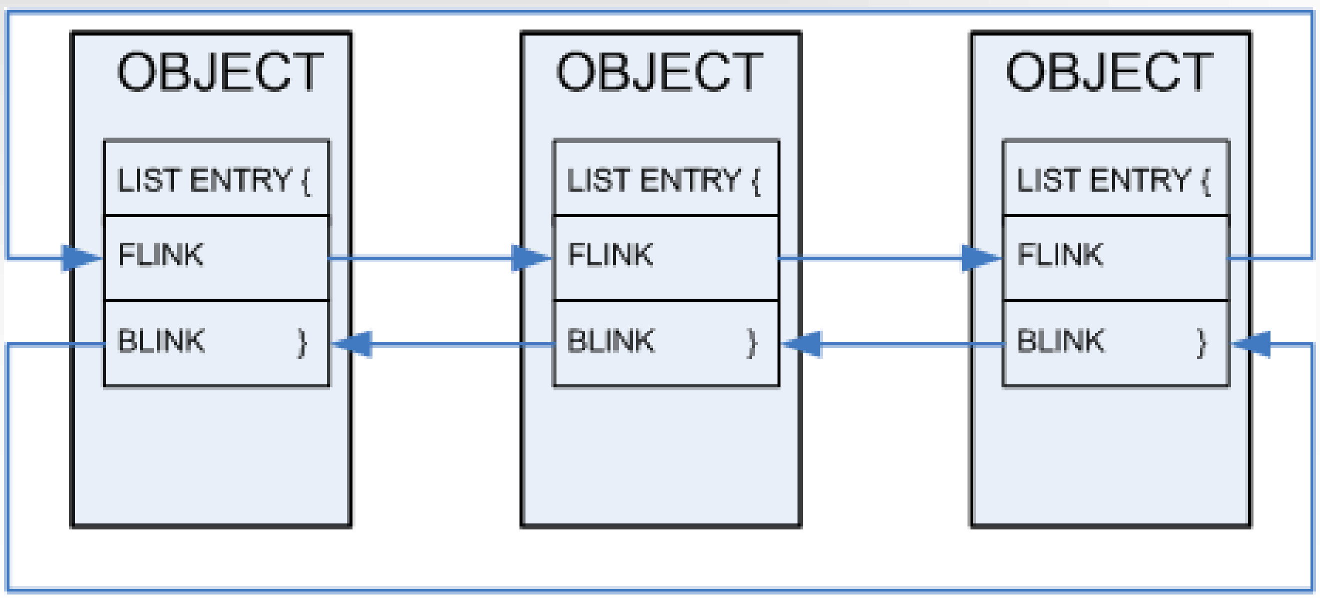
- Debug section in the ntoskrnl.exe file stores the PsInitialSystemProcess symbol



- PsInitialSystemProcess = 0x4DE000 + 0x90EF4 (RVA) = (PA) 0x56EEF4
- 0x56EEF4 -> _EPROCESS (System)

Doubly Linked List

- EPROCESS
- MODULE_ENTRY
- etc

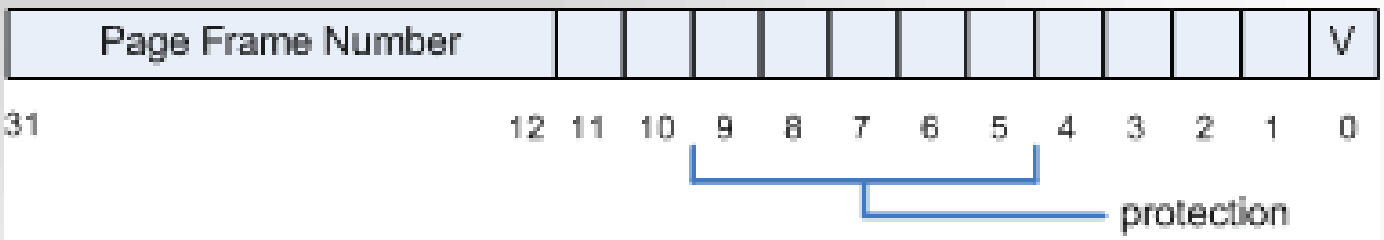


Processes' details

- SID of process owner inside ACCESS_TOKEN
- CreationTime in EPROCESS
  - *KeQuerySystemTime* is called to save the Process's Create Time
  - System time is a count of 100-nanosecond intervals since January 1, 1601. This value is computed for the GMT time zone.

Dumping memory mapped files

- Data Section Control Area
- Page Tables



- PFN * 0x1000 (Page size) = Physical Address
- Page Table entries contain index numbers to swapped-out pages when the last-significant bit is cleared
  - ➢ Index number * 0x1000 = swapped-out page frame
- Example:
  - ➢ dd.exe if=c:\memorydump.img of=page4C41 bs=4096 count=1 skip=19521 (0x4C41)

String searches

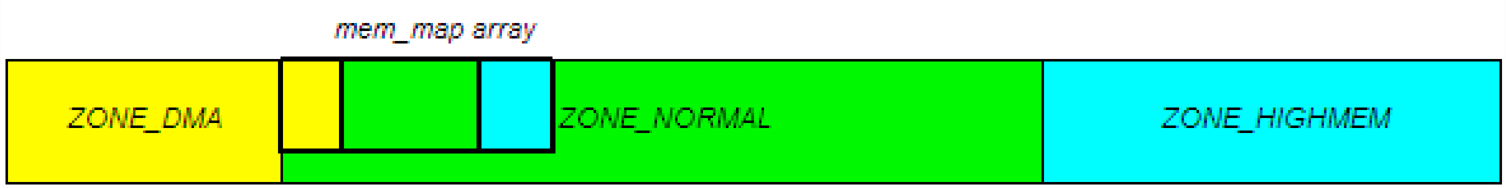
- Any tool for searching of ANSI and UNICODE strings in binary images
  - Example: Strings from Sysinternals or WinHex
- Identifying process which includes suspicious content
  - Finding PFN of Page Table which points to page frame which stores the string
  - Finding Page Directory which points to PFN of Page Table

Linux memory analysis

- Information about the analyzed memory image
  - The size of a page = 0x1000 bytes
  - The total size of the physical memory < 896 MB
  - Architecture 32-bit/64-bit/multi-threading support
- Memory layout
  - Virtual Address Space/Physical Address Space
  - User/Kernel land (3GB/1GB by default)
    - Kernel offset (PAGE_OFFSET) at 0xc0000000
  - ZONES
  - Memory map array 0xc1000030
- Knowledge about internal structures is required

Zones and Memory Map array

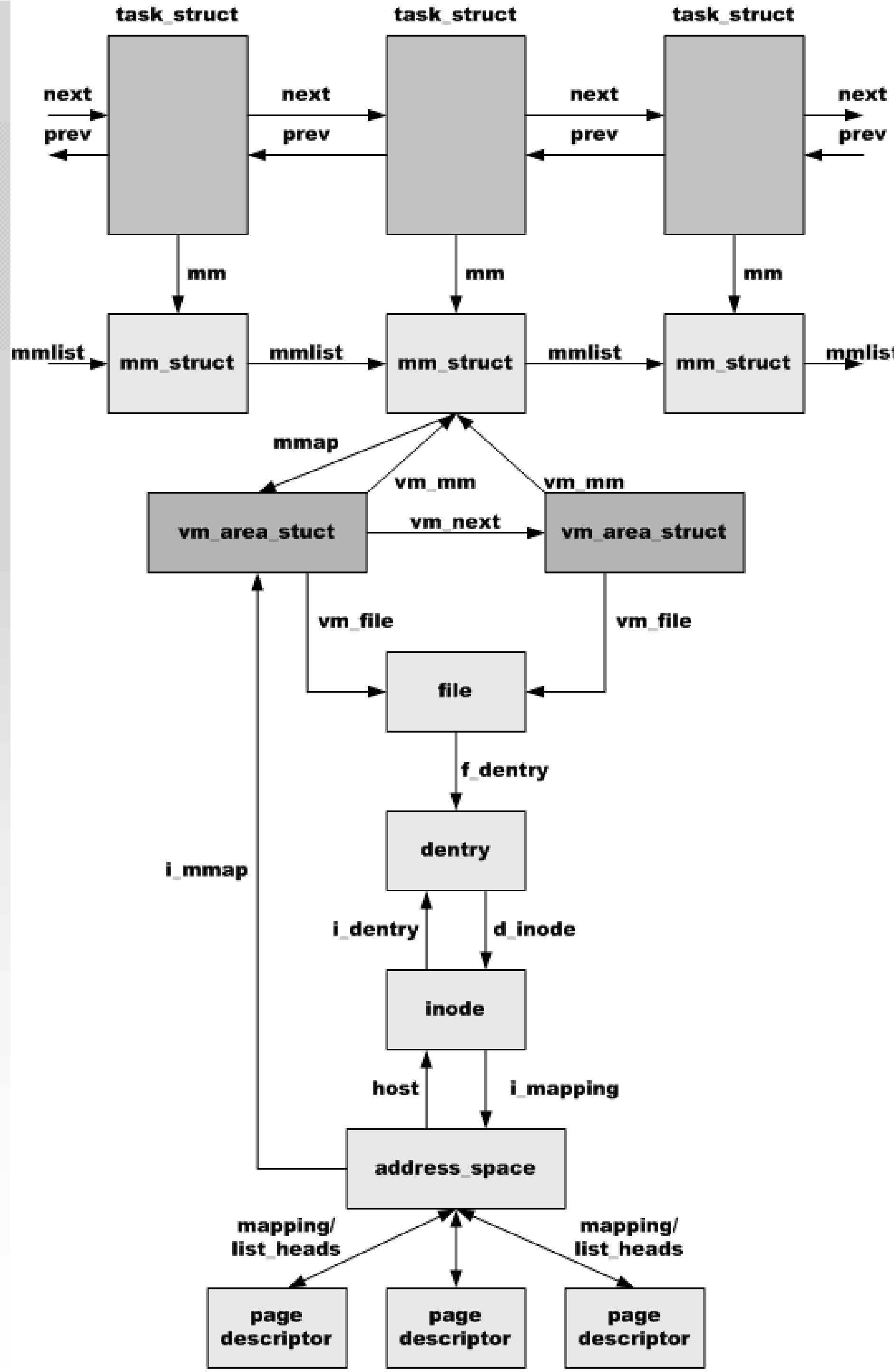
- Physical memory is partitioned into 3 zones:
  - ZONE_DMA = 16 MB
  - ZONE_NORMAL = 896 MB – 16 MB
  - ZONE_HIGHMEM > 896 MB
- The mem_map array at 0xC1000030 (VA)



Important kernel structures

- task_struct structure
- mm_struct structure
- vm_area_struct structure
- inode & dentry structures
- address_space structure
- Page descriptor structure
- mem_map array
- Page frames
  - PAGE DIRECTORY, PAGE MIDDLE DIRECTORIES & PAGE TABLES

Relations between structures



Enumerating processes

- init_task_union (process number 0)
  – The address is exported by a kernel image file
  – The address is available in the System.map file
- init_task_union struct contains list_head structure
- All processes (task_structs) are linked by a doubly linked list
- Virtual To Physical Address Translation
  ➢ VA – PAGE_OFFSET = PA

Dumping memory mapped files (e.g. process image)

- Many Incident Response Toolkits use the ptrace() function to dump a process memory
- Ptrace() based tools: memfetch, pcat, gdb, memgrep, etc…
- Each process may be only attached by one parent process
- Simple LKM:

```
…
task_lock (current);
current->ptrace=1;
task_unlock(current);
…
```

Examples:

[root@linux]# ./memgrep -p 9111 -d -a text -l 100

ptrace(ATTACH): Operation not permitted

memgrep_initialize(): Couldn't open medium device.

[root@linux bin]# ./pcat 9111

./pcat: ptrace PTRACE_ATTACH: Operation not permitted

Dumping memory mapped files (e.g. process image)

- An address_space struct points to all page descriptors
- Page descriptor
  - 0x0 –> list_head struct //doubly linked list
  - 0x8 –> mapping //pointer to an address_space
  - 0x14 –> count //number of page frames
  - 0x34 –> virtual //physical page frame

next page descriptor
address_space

```
0x010abfd8: 0xc1074278 0xc29e9528 0xc29e9528 0x00000001
0x010abfe8: 0xc1059c48 0x00000003 0x010400cc 0xc1095e04
0x010abff8: 0xc10473fc 0x03549124 0x00000099 0xc1279fa4
0x010ac008: 0xc3a7a300 0xc3123000
```

(virtual - 0xc0000000) = PA

- Flags to reduce results (e.g. VM_READ, VM_EXEC, VM_EXECUTABLE) – a vm_flags field

➢ dd if=memorydump.img of=page3123 bs=1 count=4096 skip=51523584

Finding „terminated” files (e.g. process image)

- Enumerating all page frames
  - 0x01000030 (PA)
- Fields of page descriptors are not cleared completely
  - a mapping field points to an address_space struct
  - a list_head field contains pointers to related page descriptors
- Useful information from an address_space struct
  - an i_mmap field is cleared
  - all linked page frames (clean, dirty and locked pages)
  - a host field points to an inode structure which, in turn, points to a dirent structure

Correlation with Swap Space (swap space and memory analysis)

- A mm_struct contains a pointer to the Page Global Directory (the pgd field)
- The Page Global Directory includes the addresses of several Page Middle Directories
- Page Middle Directories include the addresses of several Page Tables
- Page Table entries contain index numbers to swapped-out pages when the last-significant bit is cleared
- The first page (index 0) of the swap space is reserved for the swap header
  - (Index number x 0x1000) + 0x1000 = swapped-out page frame

Memory analysis of a live system

- Analysis of physical memory on a live system can be used to detect system compromises
- Reading kernel structures directly
  - Defeating all methods based on hijacking system calls and on modifying various tables (e.g. IDT, SDT)
  - But some functions (i.e. sys_read()) can be hooked or cheated
    - Example: Shadow Walker, the FU rootkit component, is used to defeat virtual memory scanners
  - Moreover, Direct Kernel Object Manipulation (DKOM) technique defeats a method of reading internal kernel structures directly

Finding objects hidden by DKOM

- Methods
  - Reading internal kernel structures which are not modified by rootkits
    - For example, instead of reading the list of linked EPROCESS blocks, PsActiveProcessList, we read lists of kernel threads
  - Correlating data from page frames
    - Elegant method of detecting hidden data
- 2 examples
  - Detecting hidden processes on Windows
  - Detecting hidden processes on Linux

Windows hidden processes detection

- We enumerate all linked EPROCESS blocks and store addresses of each EPROCESS block
- Next, we enumerate all entries in the PFN database and read two fields:
  - Forward link – linked page frames
  - PTE address – virtual address of the PTE that points to this page
- PTE address is in system address space and is equal to 0xC0300C00 (VA)
- Forward link points to the address of EPROCESS block
- Finally, diff-based method is used to compare a result with the doubly linked list of EPROCESS blocks

Linux hidden processes detection

- We enumerate all linked task_struct structures and store addresses of each mm_struct
- Each User Mode process has only one memory descriptor
- Next, we enumerate all page descriptors and select only page frames with memory mapped executable files (the VM_EXECUTABLE flag)
- Relations:
  - The mapping filed of a page descriptor points to the address_space struct
  - The i_mmap field of an address_space structure points to a vm_area_struct
  - The vm_mm field of a vm_area_struct points to memory descriptor
- Diff-based method is used to compare results

Integrity checks (file system and memory analysis)

- Verifying integrity of memory dump (important OS elements)
  - values stored in internal kernel tables (e.g. SCT)
  - code sections (read-only)
    - kernel image file from file system
    - other important system files from file system
- Example: kcore dump against vmlinux kernel image (from FS)

```
#gdb vmlinux kcore.image
 (gdb) disass sys_read
Dump of assembler code for function sys_read:
0xc013fb70 <sys_read>:        mov    $0xc88ab0a6,%ecx
0xc013fb73 <sys_read+3>:      jmp    *%ecx
0xc013fb77 <sys_read+7>:      mov    %esi,0x1c(%esp,1)
...
```

```
#gdb vmlinx
 (gdb) disass sys_read
Dump of assembler code for function sys_read:
0xc013fb70 <sys_read>:        sub    $0x28,%esp
0xc013fb73 <sys_read+3>:      mov    0x2c(%esp,1),%eax
0xc013fb77 <sys_read+7>:      mov    %esi,0x1c(%esp,1)
...
```

Conclusions

- Memory analysis as an integral part of Forensic Analysis
- Evidence found in a physical memory can be used to reconstruct crimes:
  - Temporal (when)
  - Relational (who, what, where)
  - Functional (how)
- Must be used to defeat anti-forensic techniques
- Can be useful in detecting system compromises on a live system

References

- Daniel P. Bovet, Marco Cesati „Understanding the Linux Kernel, 2nd Edition”
- Mark E. Russinovich, David A. Solomon, „Microsoft Windows Internals, Fourth Edition: Microsoft Windows 2003, Windows XP, and Windows 2000”
- Documents & tools at http://forensic.seccure.net

DEMO

Q & A