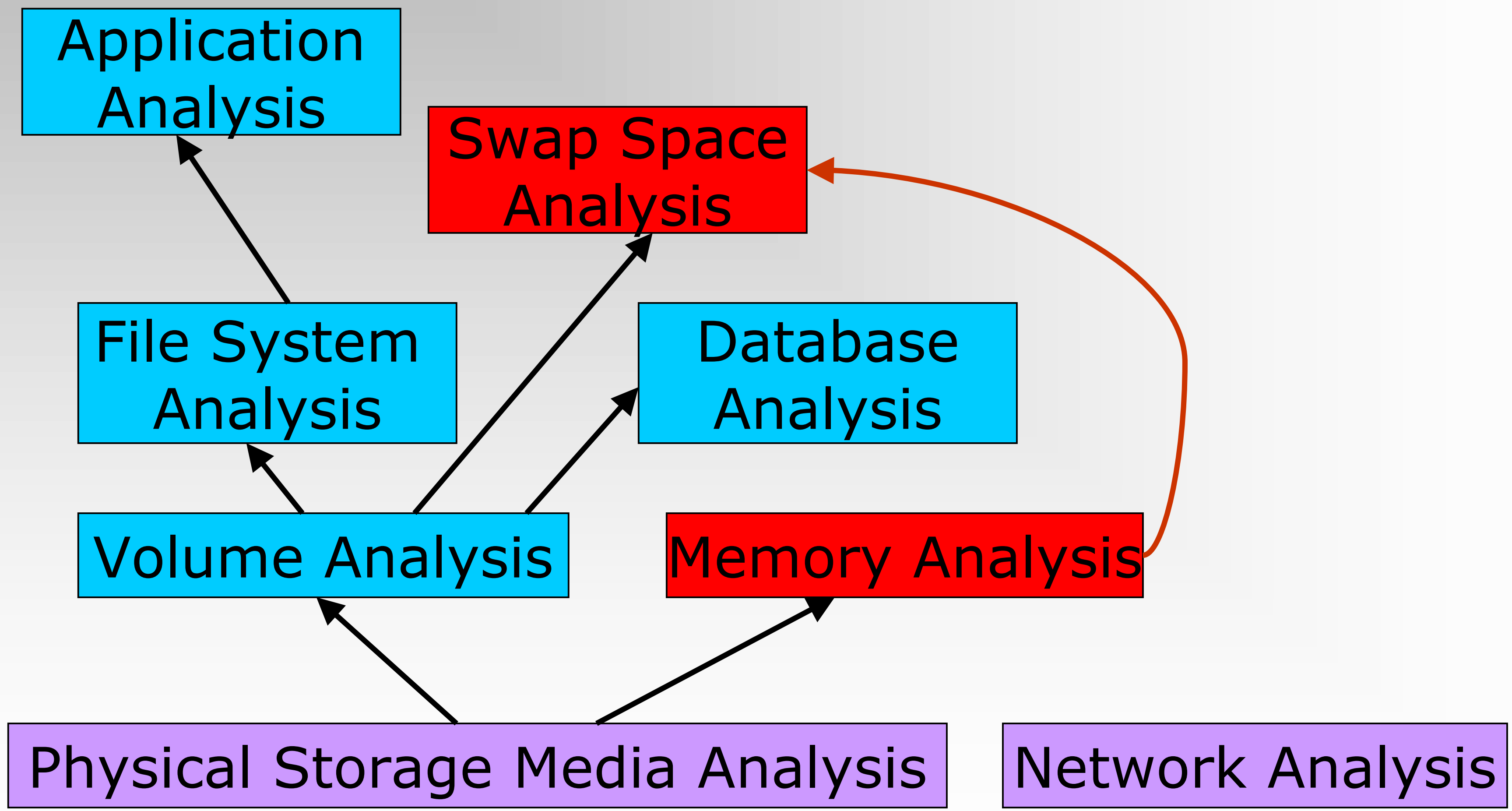
Physical Memory Forensics

Mariusz Burdach

Overview

- Introduction
- Anti-forensics
- Acquisition methods
- Memory analysis of Windows & Linux
  - Recovering memory mapped files
  - Detecting hidden data
  - Verifying integrity of core memory components
- Tools
- Q & A

Analysis Types



Source: „File System Forensic Analysis", Brian Carrier

RAM Forensics

- Memory resident data
- Correlation with Swap Areas
- Anti-Forensics against the data:
  - Data contraception
  - Data hiding
  - Data destruction
- Anti-Forensic methods:
  - Data contraception against File System Analysis
  - Data hiding against Memory Analysis

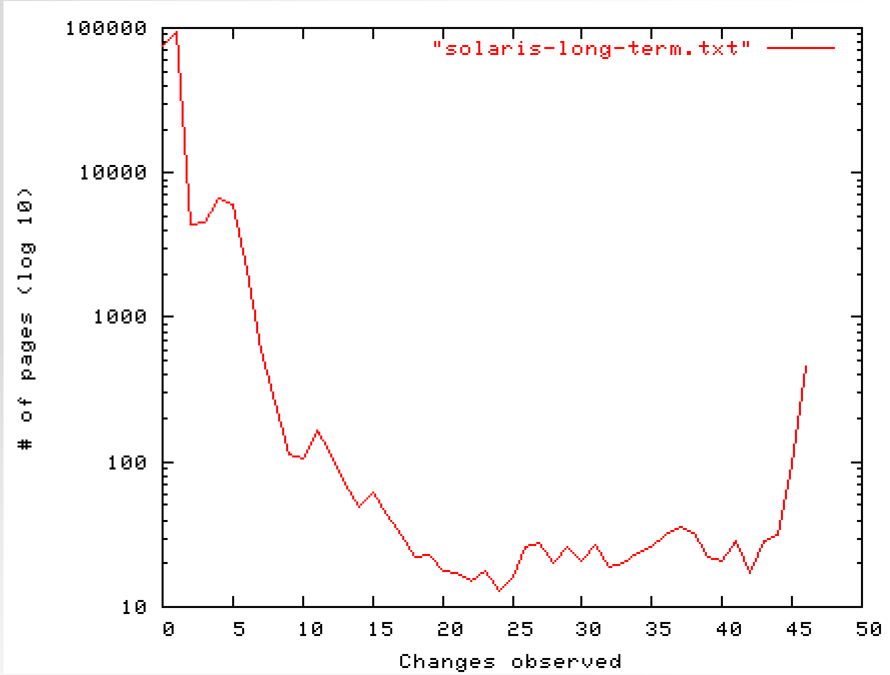
In-memory data

- Current running processes and terminated processes
- Open TCP/UDP ports/raw sockets/active connections
- Memory mapped files
  - Executable, shared, objects (modules/drivers), text files
- Caches
  - Web addresses, typed commands, passwords, clipboards, SAM database, edited files
- Hidden data and many more
- DEMO

Klip wideo

Persistence of Data in Memory

- Factors:
  - System activity
  - Main memory size
  - Data type
  - Operating system



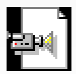
Above example*: Long-term verification of DNS server: (OS: Solaris 8, RAM: 768 MB)

Method: Tracking page state changing over time.

Result: 86 % of the memory never changes.

*Source: „Forensic Discovery", Dan Farmer, Wietse Venema

Anti-forensics

- Syscall proxying - it transparently „proxies” a process’ system calls to a remote server:
  - CORE Impact 
    Obraz - mapa bitowa
- MOSDEF - a retargetable C compiler, x86 assembler & remote code linker
  - Immunity CANVAS
- In-Memory Library Injection – a library is loaded into memory without any disk activity:
  - Metasploit’s Meterpreter (e.g. SAM Juicer)
  - DEMO 
    Klip wideo

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)
  - DEMO

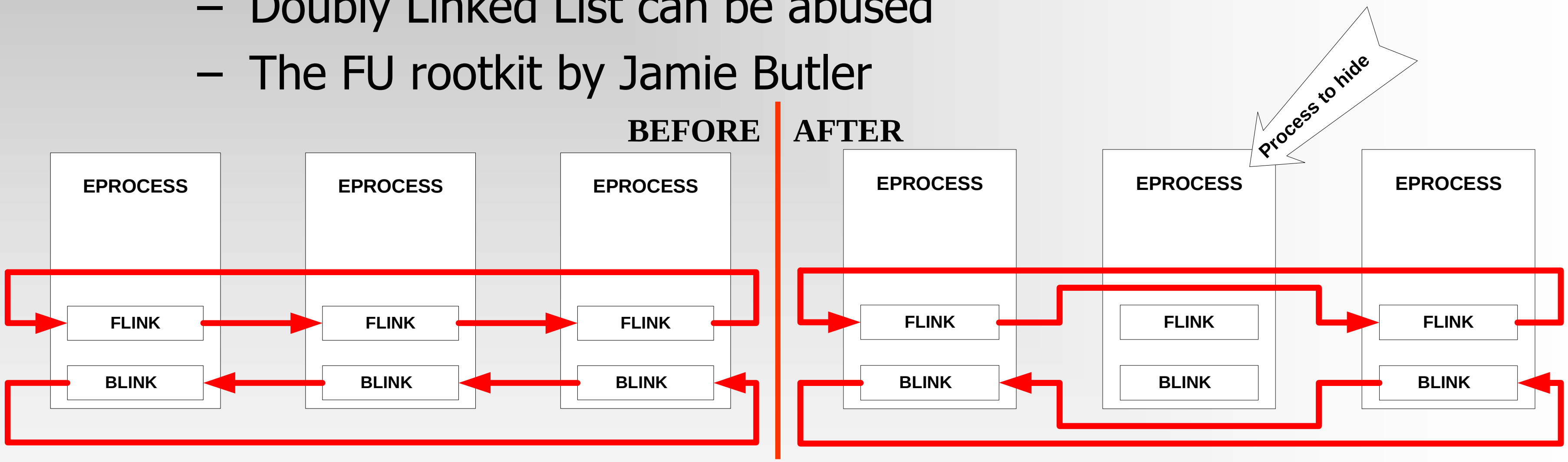
    Klip wideo
- In memory worms/rootkits
  - Their codes exist only in a volatile memory and they are installed covertly via an exploit
  - Example: Witty worm (no file payload)

Anti-forensics

- Hiding data in memory:
  - Advanced rootkits
    - Evidence gathering or incident response tools can be cheated
    - Examples:
      - Hacker Defender/Antidetection – suspended
      - FUTo/Shadow Walker
  - Offline analysis will defeat almost all methods

Anti-forensics

- DKOM (Direct Kernel Object Manipulation)
  - Doubly Linked List can be abused
  - The FU rootkit by Jamie Butler



  - Examples: Rootkit technologies in the wild*

    Worms that uses DKOM & Physical Memory:
    - W32.Myfip.H@mm
    - W32.Fanbot.A@mm

*Source: „Virus Bulletin" December, 2005, Symantec Security Response, Elia Florio

Identifying anti-forensic tools in memory image

- AF tools are not designed to be hidden against Memory Analysis
  - Meterpreter
    - Libraries are not shared
    - Server: metsrv.dll
    - Libraries with random name ext??????.dll
  - SELF
    - Executed in memory as an additional process – memory mapped files can be recovered even after process termination

Acquisition methods

- 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:
  - Hardware-based
  - Software-based

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

Software-based method

- Software-based memory acquisitions:
  - A trusted toolkit has to be used to collect volatile data
    - DD for Windows - Forensic Acquisition Utilities & KNTDD are available at http://users.erols.com/gmgarner/
    - DD for Linux by default included in each distribution (part of GNU File Utilities)
  - 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

Linux Physical memory device

- **/dev/mem** – device in many Unix/Linux systems (RAW DATA)
- **/proc/kcore** – some pseudo-filesystems provides access to a physical memory through /proc
  - This format allows us to use the gdb tool to analyse memory image, but we can simplify tasks by using some tools

Windows Physical memory device

- **\\.\PhysicalMemory** - device object in Microsoft Windows 2000/2003/XP/VISTA (RAW DATA)
- **\\.\DebugMemory** - device object in Microsoft Windows 2003/XP/VISTA (RAW DATA)
- Simple software-based acquisition procedure
  - dd.exe if=\\.\PhysicalMemory of=\\<*remote_share*>\memorydump.img
- Any Windows-based debugging tool can analyse a physical memory „image” after conversion to Microsoft crashdump format
  - http://computer.forensikblog.de/en/2006/03/dmp_file_structure.html

Problems with Software-based method

- An attacker can attack the tool
  - Blocking access to pages which are mapped with different memory types
    http://ntsecurity.nu/onmymind/2006/2006-06-01.html
- Problems with access to a physical memory from user level
  - Windows 2003 SP1+ & Vista
  - Linux
    - SYS_RAWIO capability of Capability Bounding Set
  - It is vital to use kernel driver

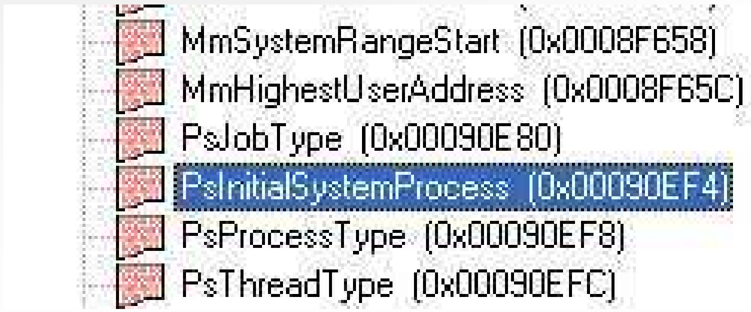
Why physical backup is better?

- Limitations of logical backup
  - Partial information
    - selected data
    - only allocated memory
  - Rootkit technologies
  - Many memory and swap space modification
- Incident Response (First Response) Systems
  - Set of tools
    - Forensic Server Project
    - Foundstone Remote Forensics System
  - Direct calls to Windows API
    - FirstResponse - Mandiant
    - EnCase Enterprise Edition
  - Cheating IR tools (DEMO)

Klip wideo

Preparation

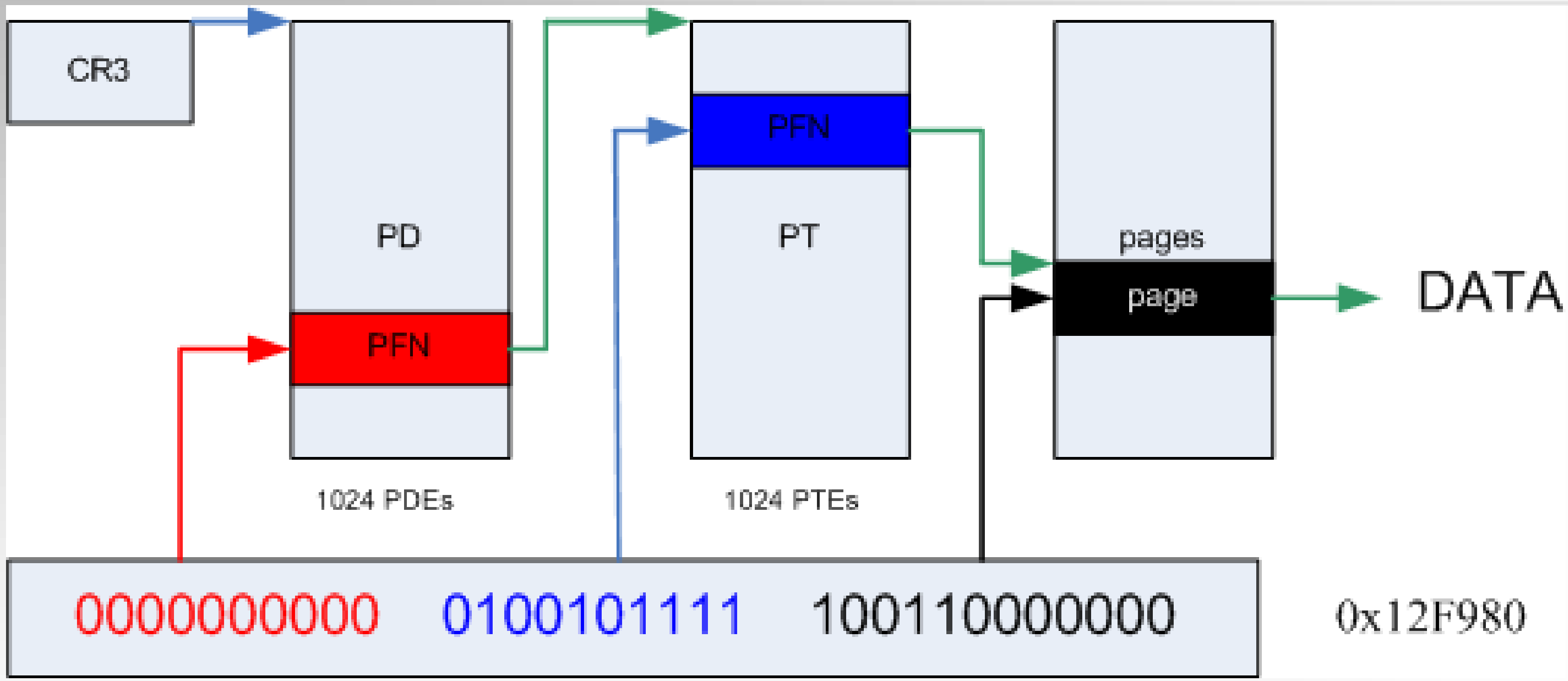
- Useful files (acquired from a file system):
  - Kernel image files (ntoskrnl.exe, vmlinux-2.x)
  - Drivers/modules/libraries
  - Configuration files (i.e. SAM file, boot.ini)
- These files must be trusted
  - File Hash Databases can be used to compare hash sums
- Map of Symbols
  - System.map file
  - Some symbols are exported by core operating system files



System identification

- Information about the analysed memory dump
  - The size of a page =4096 (0x1000) bytes
  - The total size of the physical memory
    - Physical Address Extension (PAE)
    - HIGHMEM = 896 MB
  - Architecture 32-bit/64-bit/IA-64/SMP
- Memory layout
  - Virtual Address Space/Physical Address Space
  - User/Kernel land
    - Windows kernel offset at 0x80000000
    - Linux kernel offset at 0xC0000000
  - (Windows) The PFN Database at 0x80C00000
  - (Linux) The Mem_Map Database at 0xC1000030
  - (Windows) The PTE_BASE at 0xC0000000 (on a non-PAE systems)
  - Page directory – each process has only one PD
- Knowledge about internal structures is required

Virtual ->Physical (x86)



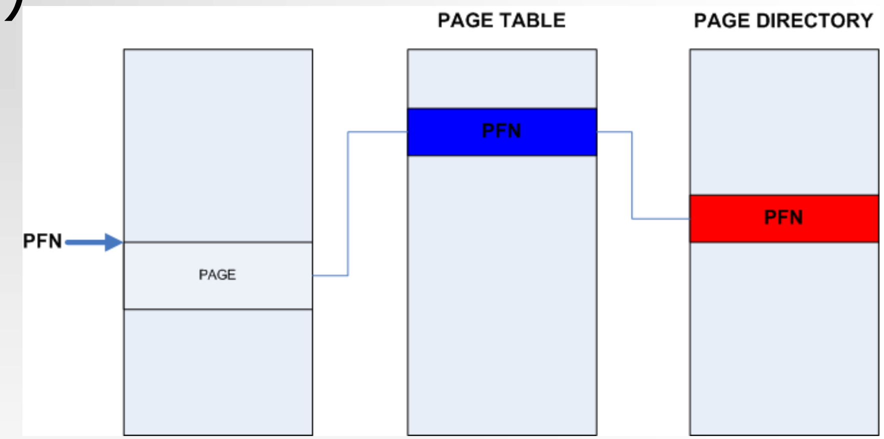
(Windows) PTE address = PTE_BASE + (page directory index) * PAGE_SIZE

+ (page table index) * PTE size

(Linux) PA = VA – PAGE_OFFSET

Physical ->Virtual (x86)

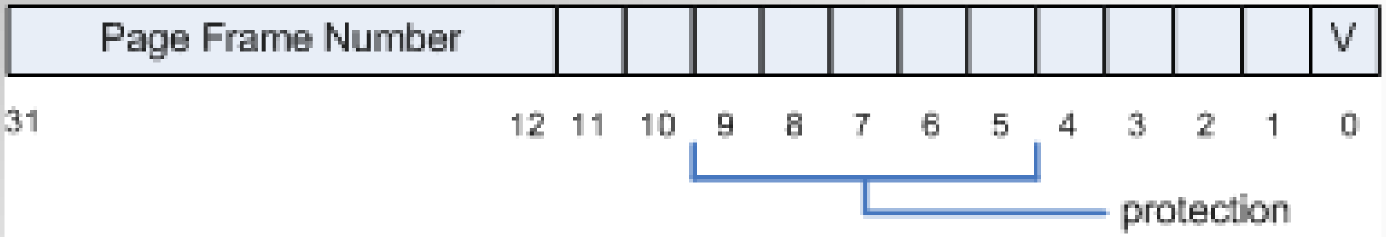
- PFN & mem_map databases
- Entries represent each physical page of memory on the system (not all pages!)




```
PFN 000263A3 at address 813D8748
flink 000002D4  blink / share count 00000001  pteaddress E42AF03C
reference count 0001   Cached     color 0
restore pte F8A10476  containing page  02597C  Active     P
Shared
```

Page Table Entries

- Page Table Entry



- There are PAGE_SHIFT (12) bits in 32-bit value that are free for status bits of the page table entry
- PTE must be checked to identify the stage of a page
- PFN * 0x1000 (Page size) = Physical Address

Correlation with Swap Space

- Linux: A mm_struct contains a pointer to the Page Global Directory (the pgd field)
- Windows: A PCB substructure contains a pointer to the Directory Table Base
- Page Table entries contain index numbers to swapped-out pages when the last-significant bit is cleared
  - Linux: (Index number x 0x1000 (swap header)) + 0x1000 = swapped-out page frame
  - Windows: Index number x 0x1000 = swapped-out page frame

Methods of analysis

- Strings searching and signatures matching
  - extracting strings from images (ASCII & UNICODE)
  - identifying memory mapped objects by using signatures (e.g. file headers, .text sections)
- Interpreting internal kernel structures
- Enumerating & correlating all page frames

Strings & signatures searching

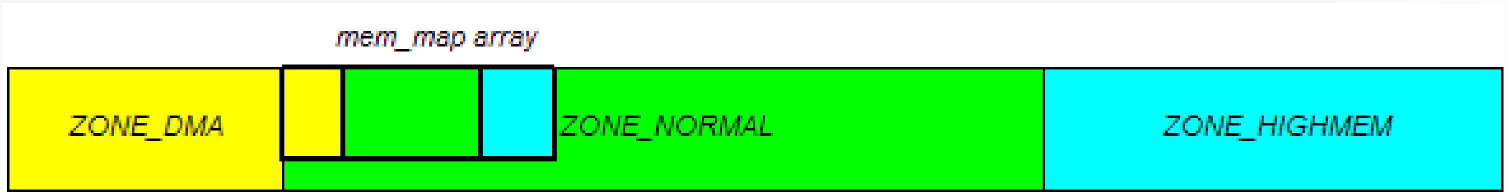
- Any tool for searching of ANSI and UNICODE strings in binary images
  - Example: Strings from Sysinternals or WinHex
- Any tool for searching of fingerprints in binary images
  - Example: Foremost
- Identifying process which includes suspicious content:
  - Finding PFN of Page Table which points to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

Klip wideo

LINUX internal structures

Zones and Memory Map array

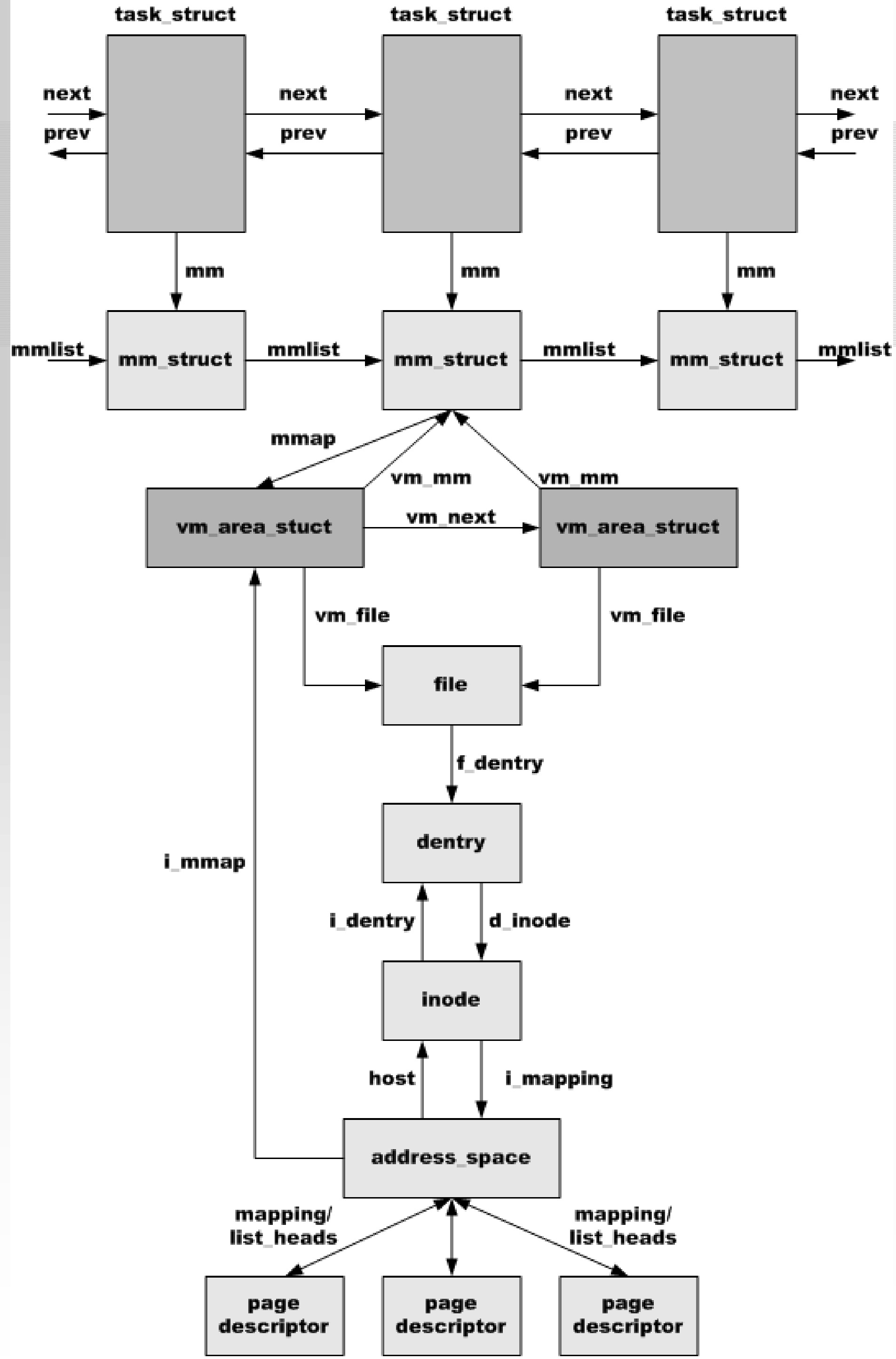
- Physical memory is partitioned into 3 zones:
  - ZONE_DMA = 16 MB
  - ZONE_NORMAL = 896 MB – 16 MB
  - ZONE_HIGHMEM > 896 MB
- The mem_map array at 0xC1000030 (VA)



Important kernel structures

- task_struct structure
  - mm_struct structure
  - vm_area_struct structure
  - inode & dentry structures – e.g. info about files and MAC times
  - address_space structure
- mem_map array
  - Page descriptor structure

Relations between structures

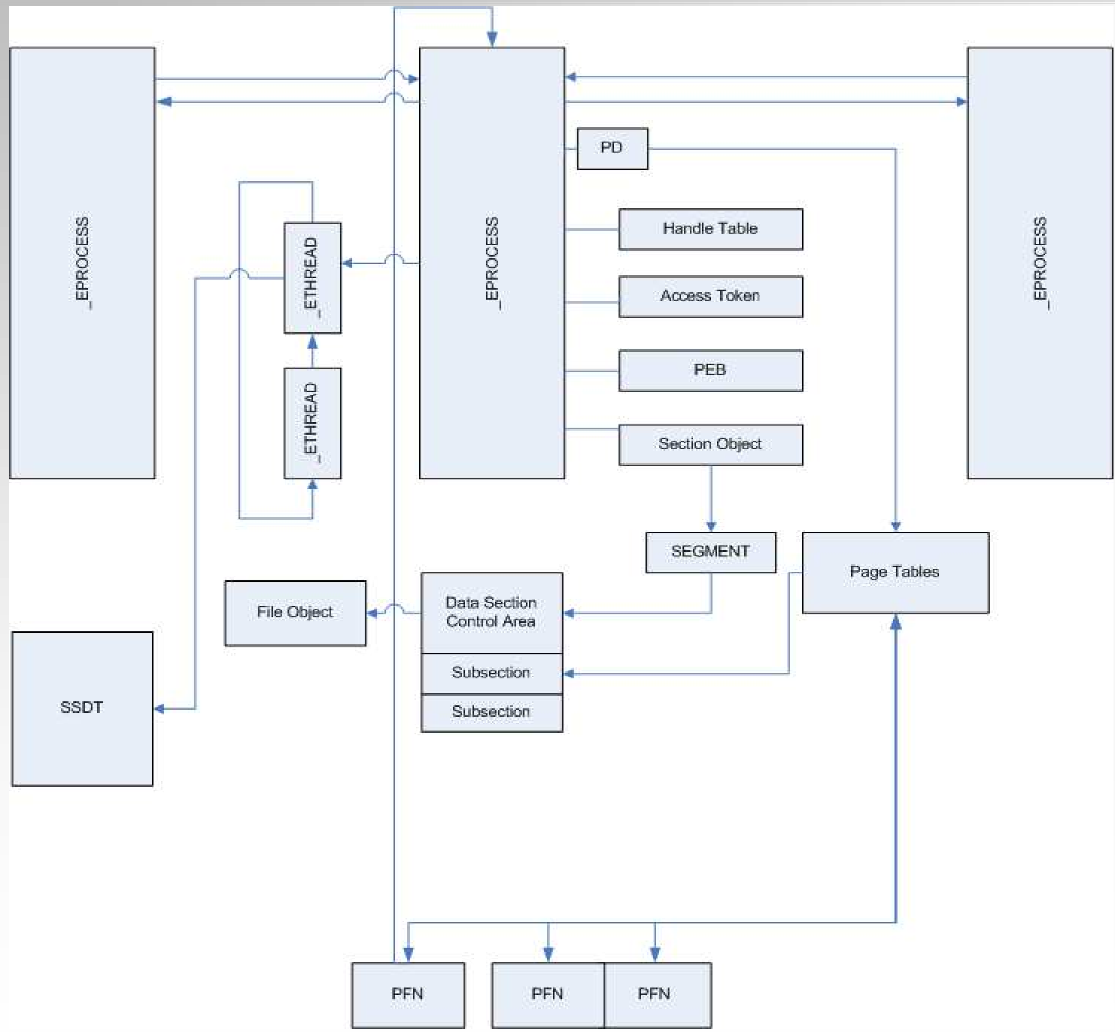


Windows internal structures

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
  - CreationTime - a count of 100-nanosecond intervals since January 1, 1601
  - Data Section Control Area
    - Page frames
- PFN (Page Frame Number) Database
  - PFN entries

Relations between structures



Enumerating processes

- Linux
  - init_task_union (process number 0)
    - The address is exported by a kernel image file
    - The address is available in the System.map file
    - String searches method
  - init_task_union struct contains list_head structure
  - All processes (task_structs) are linked by a doubly linked list
- Windows
  - PsInitialSystemProcess (ntoskrnl.exe) = _EPROCESS (System)
  - _EPROCESS blocks are linked by a doubly linked list

Linux: Dumping memory mapped files

- Page Tables to verify the stage of pages
- 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

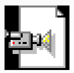
address_space

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

(virtual - 0xc0000000) = PA

Linux: Dumping memory mapped files

- Signature (strings or hex values) searching
- Reconstructing objects:
  - Finding page descriptor which points to page frame which stores the signature (mem_map array)
  - Page descriptor points to all related page descriptors (the sequence is critical)
  - We have all page frames and size of file (inode structure)
- DEMO



Klip wideo

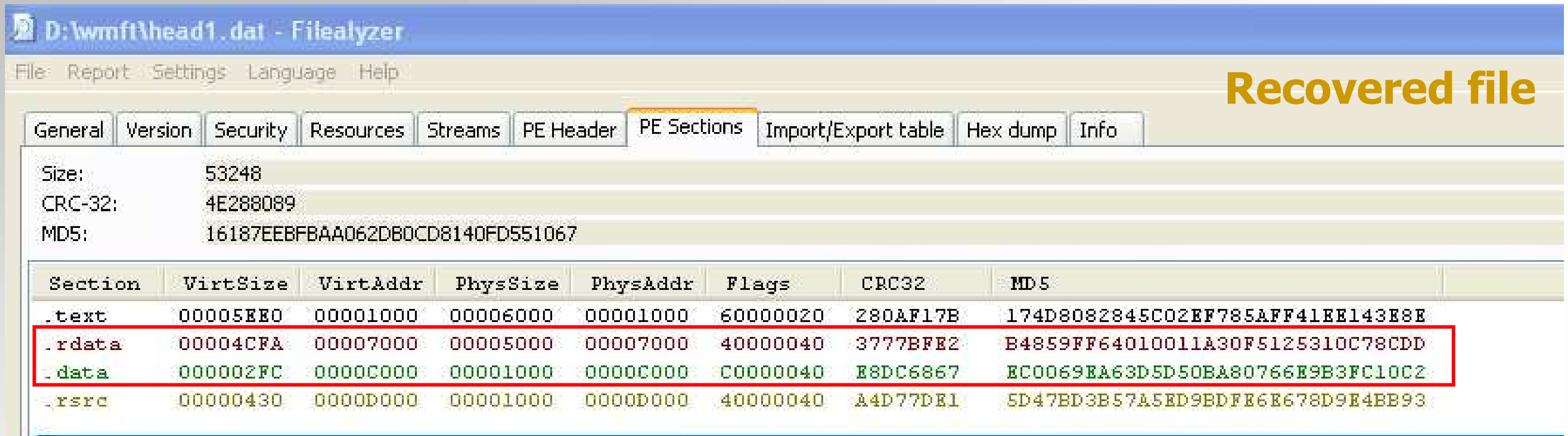
Windows: Dumping memory mapped files

- Page Tables to check the stage of pages
- Data Section Control Area
- Information from the first page (PE header)
  - PEB -> ImageBaseAddress

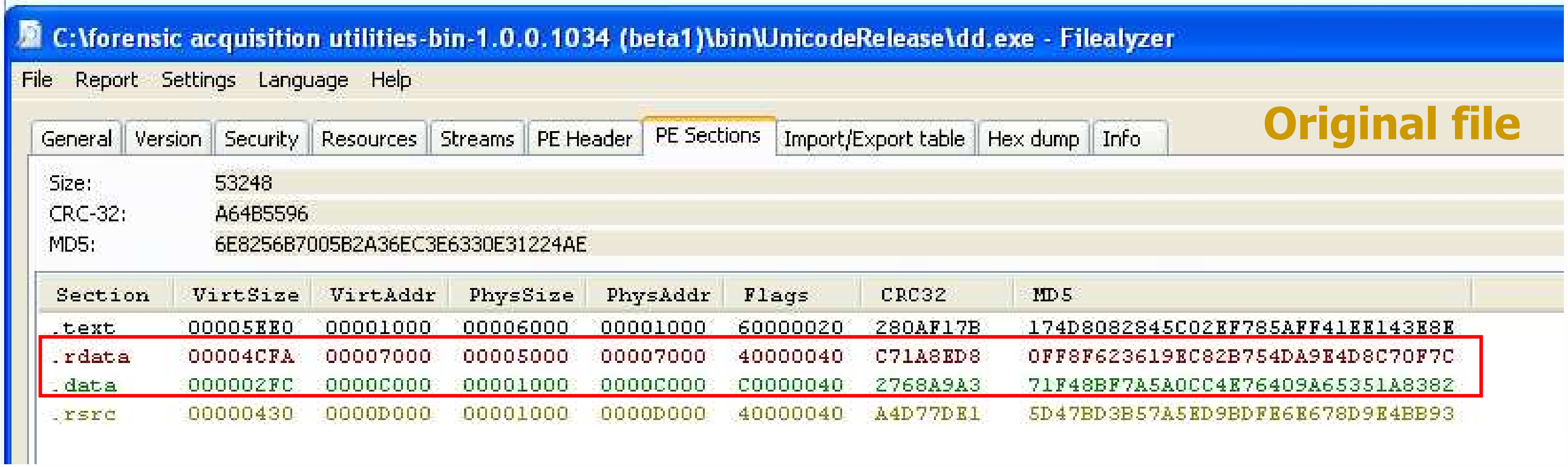
Name	Virtual Size	Virtual Address	Size of Raw Data	Pointer to Raw Data
.text	00005EE0h	00401000h	00006000h	00001000h
.rdata	00004CFAh	00407000h	00005000h	00007000h
.data	000002FCh	0040C000h	00001000h	0000C000h
.rsrc	00000430h	0040D000h	00001000h	0000D000h

- Required information:
  - the Page Directory of the Process (for dumping process image file)
  - the Page Directory of the System process (for dumping drivers/modules)

Integrity verification



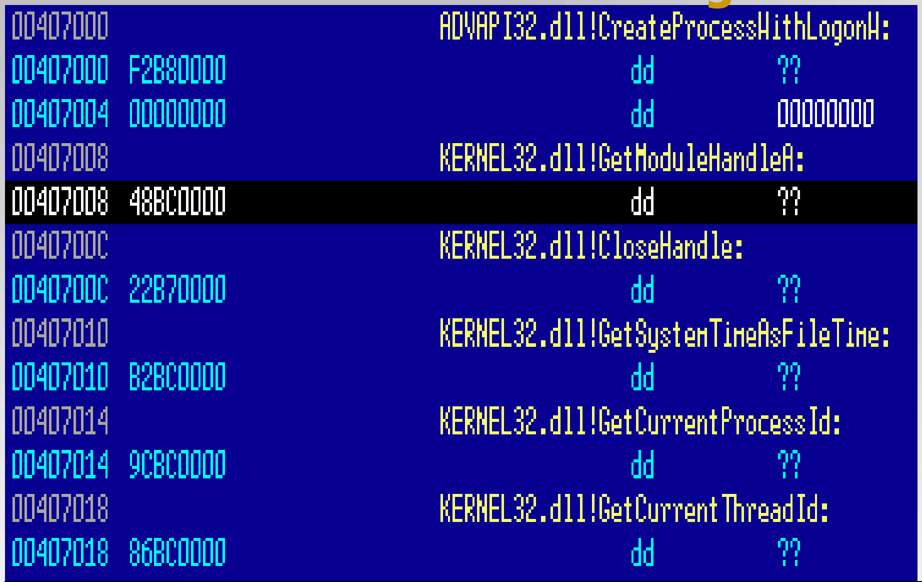
Section	VirtSize	VirtAddr	PhysSize	PhysAddr	Flags	CRC32	MD5
.text	00005EE0	00001000	00006000	00001000	60000020	280AF17B	174D8082845C02EF785AFF41EE143E8E
.rdata	00004CFA	00007000	00005000	00007000	40000040	3777BFE2	B4859FF64010011A30F5125310C78CDD
.data	000002FC	0000C000	00001000	0000C000	C0000040	E8DC6867	EC0069EA63D5D50BA80766E9B3FC10C2
.rsrc	00000430	0000D000	00001000	0000D000	40000040	A4D77DE1	5D47BD3B57A5ED9BDFE6E678D9E4BB93



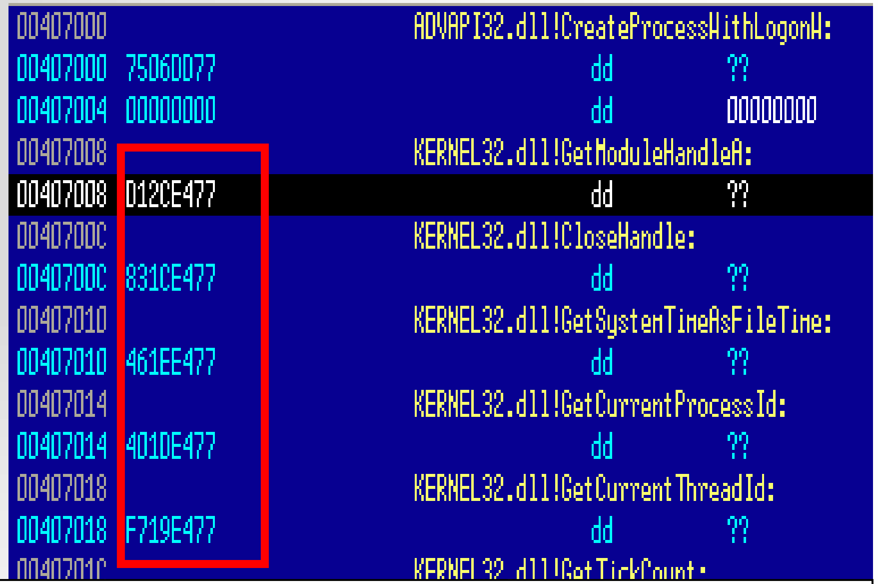
Section	VirtSize	VirtAddr	PhysSize	PhysAddr	Flags	CRC32	MD5
.text	00005EE0	00001000	00006000	00001000	60000020	280AF17B	174D8082845C02EF785AFF41EE143E8E
.rdata	00004CFA	00007000	00005000	00007000	40000040	C71A8ED8	0FF8F623619EC82B754DA9E4D8C70F7C
.data	000002FC	0000C000	00001000	0000C000	C0000040	2768A9A3	71F48BF7A5A0CC4E76409A65351A8382
.rsrc	00000430	0000D000	00001000	0000D000	40000040	A4D77DE1	5D47BD3B57A5ED9BDFE6E678D9E4BB93

IAT in .rdata

Original file



Recovered file




```
kd> u 0x77e42cd1
kernel32!GetModuleHandleA:
77e42cd1 837c240400      cmp     dword ptr [esp+0x4],0x0
77e42cd6 7418            jz      kernel32!GetModuleHandleA+0x1f (77e42cf0)
77e42cd8 ff742404        push    dword ptr [esp+0x4]
...
```

Finding hidden objects

- Methods
  - Reading internal kernel structures which are not modified by rootkits
    - List of threads instead list of processes
    - PspCidTable
    - Etc...
  - Grepping Objects
    - Objects like Driver, Device or Process have static signatures
      - Data inside object
      - Data outside object
  - Correlating data from page frames
    - Elegant method of detecting hidden data

Windows: Finding hidden objects (_EPROCESS blocks)

```
PFN 00025687 at address 813C4CA8
flink       8823A020  blink / share count 00000097  pteaddress C0300C00
reference count 0001   Cached     color 0
restore pte 00000080  containing page       025687  Active     M
Modified
```

- Enumerating PFN database
- Verifying following fields:
  - Forward link – linked page frames (Forward link also points to the address of EPROCESS block)
  - PTE address – virtual address of the PTE that points to this page
  - Containing page – points to PFN which points to this PFN
- DEMO

Klip wideo

Linux: Finding hidden objects (mm_struct structure)

- 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

Windows: Finding hidden objects (_MODULE_ENTRY)

- Scanning physical memory in order to find memory signatures
  - Identification of module header (MZ header)
  - Identification of module structures
    - Inside object – Driver Object
      GREPEXEC
      http://www.uninformed.org/?v=4&a=2
    - Outside object

```
typedef struct _MODULE_ENTRY {
 LIST_ENTRY module_list_entry;
 DWORD unknown1[4];
 DWORD base;
 DWORD driver_start;
 DWORD unknown2;
 UNICODE_STRING driver_Path;
 UNICODE_STRING driver_Name;
}
```

```
01D65190  79 00 73 00 00 00 4C 64  0E 00 0E 0A 4D 6D 4C 64  y s   Ld    MmLd
01D651A0  30 51 96 81 10 52 96 81  FF FF FF FF FF FF FF FF  0Q–□ R–□˙˙˙˙˙˙˙˙
01D651B0  00 00 00 00 00 00 00 00  00 30 81 F9 C3 F1 81 F9          0□ůĂń□ů
01D651C0  00 F0 00 00 14 00 14 00  A0 1F 00 E1 14 00 14 00   đ         á
01D651D0  EC 51 96 81 00 40 00 09  01 00 00 00 00 00 00 00  ěQ–□ @
01D651E0  DB 46 01 00 FE FF FF FF  00 00 00 00 69 00 73 00  ŰF  ţ˙˙˙    i s
01D651F0  61 00 70 00 6E 00 70 00  2E 00 73 00 79 00 73 00  a p n p . s y s
01D65200  00 00 0D 0A 4D 6D 4C 64  0E 00 0D 0A 4D 6D 4C 64      MmLd    MmLd
```

Detecting modifications of memory

- Offline detection of memory modifications
  - System call hooking
    - Function pointers in tables (SSDT, IAT, SCT, etc)
  - Detours
    - Jump instructions
- Cross-view verification
  - .text sections of core kernel components
  - values stored in internal kernel tables (e.g. SCT)

SSDT

- Verification of core functions by comparing first few bytes
  - Self-modifying kernel code
    - Ntoskrnl.exe & Hall.dll
- Finding an address of KiServiceTable
  - Memory image file: _KTHREAD (TCB)
    - *ServiceTable = 80567940
  - Symbols exported by the ntoskrnl.exe (debug section):
    - `NtAllocateUuids (0x0010176C)`
    - `NtAllocateVirtualMemory (0x00090D9D)`

SSDT in the ntoskrnl.exe

```
_text:0040B6A8 off_0_40B6A8     dd offset loc_0_4AF2DE  ; DATA XREF:
_text:0040B6AC                  dd offset loc_0_498DED
_text:0040B6B0                  dd offset loc_0_4B245B
_text:0040B6B4                  dd offset loc_0_4B0080
_text:0040B6B8                  dd offset loc_0_4BBA37
_text:0040B6BC                  dd offset loc_0_55F4D0
_text:0040B6C0                  dd offset loc_0_561661
_text:0040B6C4                  dd offset loc_0_5616AA
_text:0040B6C8                  dd offset NtAddAtom
_text:0040B6CC                  dd offset loc_0_56FECF
_text:0040B6D0                  dd offset loc_0_55EC93
_text:0040B6D4                  dd offset NtAdjustPrivilegesToken
_text:0040B6D8                  dd offset loc_0_556DD4
_text:0040B6DC                  dd offset loc_0_4A2BB8
_text:0040B6E0                  dd offset NtAllocateLocallyUniqueId
_text:0040B6E4                  dd offset loc_0_54DEFD
_text:0040B6E8                  dd offset NtAllocateUuids
_text:0040B6EC                  dd offset NtAllocateVirtualMemory
_text:0040B6F0                  dd offset loc_0_4FE30D
_text:0040B6F4                  dd offset loc_0_4C7422
_text:0040B6F8                  dd offset loc_0_40BCB4
_text:0040B6FC                  dd offset loc_0_570443
_text:0040B700                  dd offset loc_0_4EEA9C
_text:0040B704                  dd offset loc_0_423007
_text:0040B708                  dd offset loc_0_491449
_text:0040B70C                  dd offset NtClose
_text:0040B710                  dd offset loc_0_4BB42C
_text:0040B714                  dd offset loc_0_575ED5
```

Linux: removing data

- The content of page frames is not removed
- 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
- Finding „terminated” files
  – Enumerating all page frames - 0x01000030 (PA)
  – A page descriptor points to an address_space
  – 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

Windows: removing data

- The content of page frames is not removed
- All fields in PFN, PDEs & PTEs are cleared completely
- Information from related kernel structures are also cleared
- We can recover particular page frames but it is impossible to correlate them without context

Available tools

- Debugging tools (kcore & crashdump)
- Analysis of Windows memory images
  - **KNTTools** by George M. Garner Jr.
    - **KNTDD & KNTLIST**
  - **WMFT** - Windows Memory Forensics Toolkit at http://forensic.seccure.net
- Analysis of Linux memory images
  - **IDETECT** at http://forensic.seccure.net

KNTTOOLS

- KNTDD
    - MS Windows 2000SP4/XP+/2003+/Vista
    - Conversion to MS crash dump format
- KNTLIST
    - Information about system configuration
        - System Service & Shadow Service Tables
        - IDT & GDT Tables
        - Drivers & Devices Objects
        - Enumerates network information such as interface list, arp list, address object, NIDS blocks and TCB table
    - Information about processes
        - Threads, Access Tokens
        - Virtual Address Space, Working Set
        - Handle table, Executive Objects, Section Object
        - Memory Subsections & Control Area
    - References are examined to find hidden data

WMFT

- Support for Windows XP & 2003
- Functionality
  - Enumerating processes, modules, libraries (doubly linked list)
  - Finding hidden data – processes and modules (grepping objects & correlating pages)
  - Verifying integrity of functions
  - Dumping process image file and modules
  - Detailed info about processes
    - Access Token, Handle Table, Control Area & Subsections, etc
  - Enumerating & finding PFNs
- To do:
  - The disassembly functionality
  - Support for Vista

Conclusion

- Memory analysis as an integral part of Forensic Analysis
- Evidence found in physical memory can be used to reconstruct crimes:
  - Temporal (when)
  - Relational (who, what, where)
  - Functional (how)
- Sometimes evidence can be resident only in physical memory
- Must be used to defeat anti-forensic techniques

Q & A

Thank you.

Mariusz.Burdach@seccure.net

http://forensic.seccure.net