



Physical Memory Analysis

Mariusz Burdach

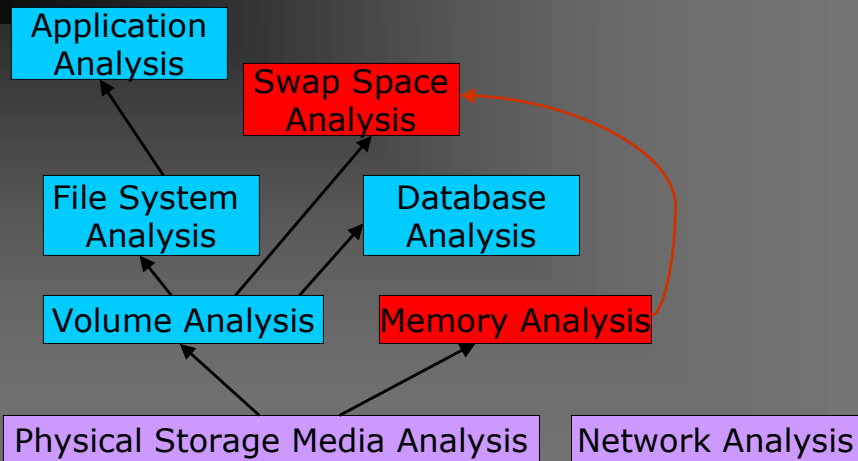
Innovations in Digital Forensic Practice
March 27-28, 2006
Washington, DC



Overview

- Introduction
- Acquisition methods
- Memory analysis
 - Linux
 - Windows
- Q & A

Analysis Types



Source: „File System Forensic Analysis“, Brian Carrier

Memory Forensics

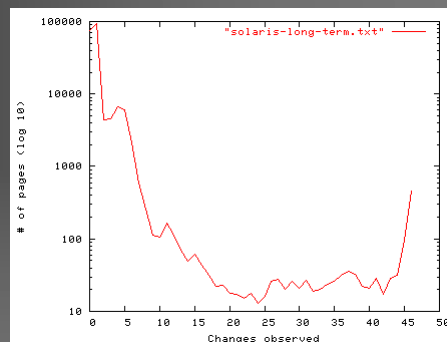
- Memory resident data
- Correlation with Swap Areas
- File System Anti-Forensics
 - Data contraception
 - Data hiding
 - Data destruction
- Hiding data in memory - offline analysis will defeat all methods of hiding objects except DKOM (Direct Kernel Object Manipulation)

In-memory data

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

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
- 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 video

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)
- 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

- Advanced rootkits
 - Evidence gathering or incident response tools can be easily cheated
 - Examples: Hacker Defender/Antidetecion, FU/Shadow Walker
- Rootkit technologies in the wild*

Worms that use DKOM & Physical Memory:

 - W32.Myfip.H@mm
 - W32.Fanbot.A@mm

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



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:
 - 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



Physical memory devices

- **\\.\PhysicalMemory** - device object in Microsoft Windows 2000/2003/XP
- **/dev/mem** – device in many Unix/Linux systems
- **/proc/kcore** – some pseudo-file systems 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/gmgamer/forensics/>
- DD for Linux by default included in each distribution (part of GNU File Utilities)



Why physical backup is better?

- Limitations of logical backup
 - Partial information
 - selected data
 - allocated memory only
 - Rootkit technologies
 - Memory and swap space modification



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



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

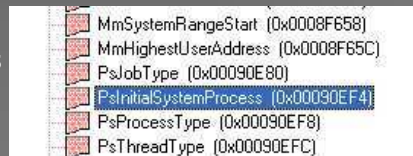


Related work (MS Windows)

- Digital Forensic Research WorkShop
- The Memory Analysis Challenge 2005
- 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)

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



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

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
- DEMO  Klip video

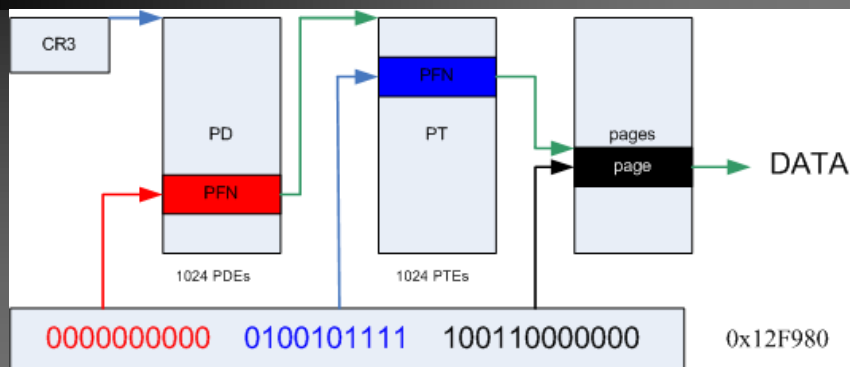
Signature matching

- Tool for searching of fingerprints in binary images
 - Example: Foremost
- 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)

System identification

- Information about the analysed memory dump
 - The size of a page = 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
 - (Windows) The PTE Base
 - (Linux) Zones
 - Page directory – each process has only one PD
- Knowledge about internal structures is required

Paging (x86)



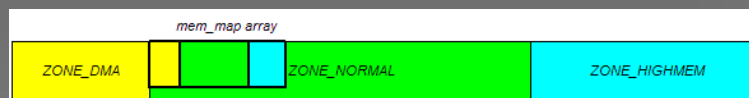
(Windows) PTE address = PTE_BASE + (page directory index) * PAGE_SIZE
+ (page table index) * PTE size

(Linux) PA = VA - PAGE_OFFSET

LINUX

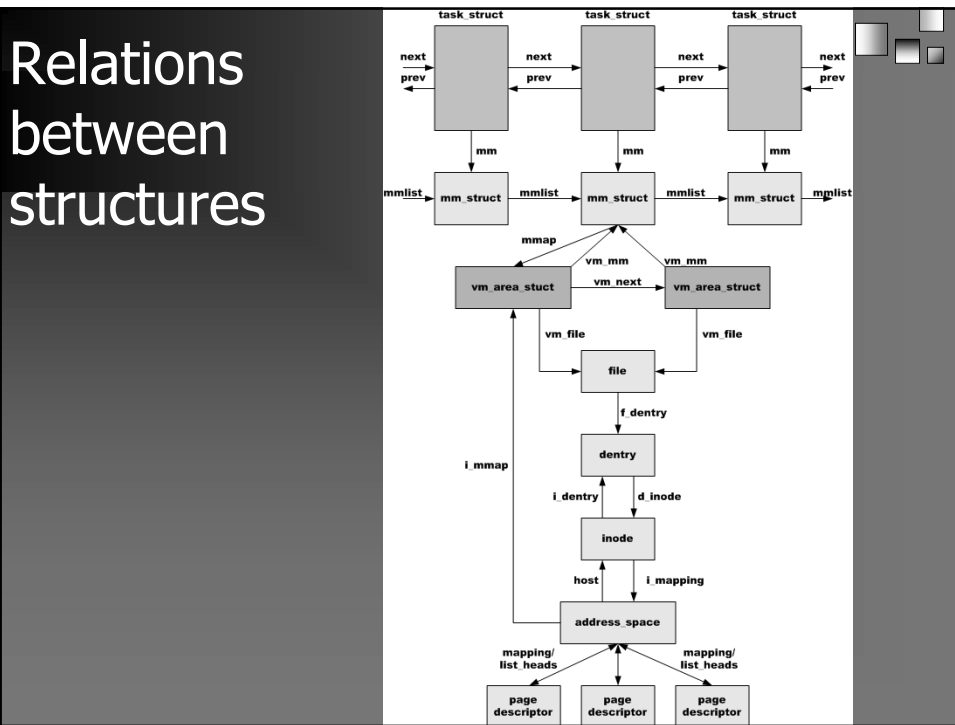
Zones and Memory Map array

- 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





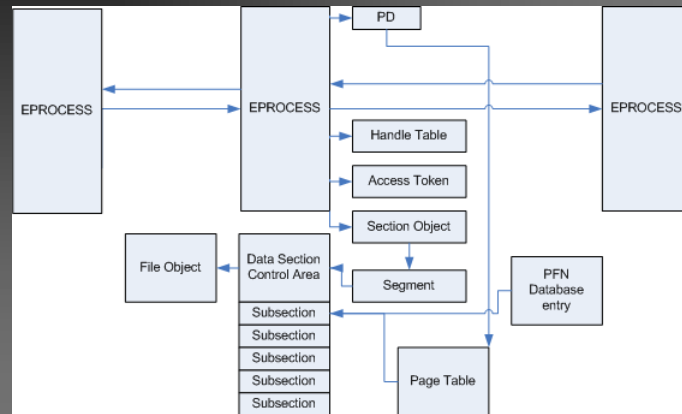
Windows



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

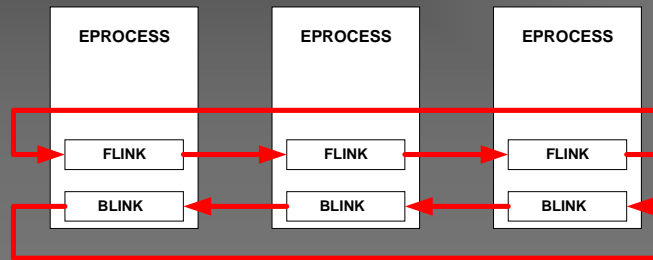


Enumerating processes

- 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

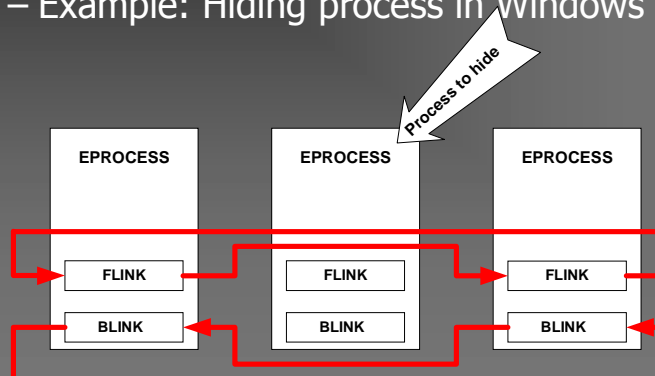
Doubly Linked List

- Processes, Modules/Drivers, etc
- Abused to the DKOM technique




Direct Kernel Object Manipulation

- The FU rootkit by Jamie Butler
– Example: Hiding process in Windows



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
- DEMO  Klip video

Linux: Dumping memory mapped files

- Page Tables
- 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
```

➤ dd if=memorydump.img of=page3123 bs=4096 count=1 skip=12579 (0x3123)





## Linux: Finding „terminated“ files

- 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



## 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“
- Dan Farmer, Wietse Venema „Forensic Discovery“
- Brian Carrier „File System Forensic Analysis“
- Documents & tools at <http://forensic.secure.net>



Q & A



Thank you.

[Mariusz.Burdach@seccure.net](mailto:Mariusz.Burdach@seccure.net)  
<http://forensic.seccure.net>