Locating Processes in Memory Images

Karla Saur &
Julian Grizzard

Sponsored By:
Wright-Patterson Air Force Base – AFRL
Johns Hopkins University Applied Physics Lab

Project Background

- Critical Challenge: Detecting Malware in Memory Images
  - Digital memory forensics allows an investigator to discover intrusions that may have occurred
  - Our research advances the understanding of malware by uncovering some of its fundamental weaknesses
- Our Approach:
  - Static memory image forensic analysis for Windows and Linux systems
  - Algorithms are anomaly based; search for malware independent of pre-existing assumptions

Memory Forensics

- Memory resident in data:
  - Processes
  - Open Sockets/Sessions
  - Memory Mapped Files
  - User Data
- Our Focus: Hidden Processes
  - Operating system keeps a list of all running processes
  - Processes hidden by malware/rootkits
  - Direct Kernel Object Manipulation – common hiding technique

Hidden Processes

- Common Method To Find Hidden Processes:
  - Search for kernel objects hidden in memory
  - Problem: kernel objects are OS specific; may be unpredicably altered
- Our Alternative Method:
  - Bring the search down to the hardware layer and search for x86 paging structures in memory
  - Hardware layer is OS independent and much more difficult to tamper with

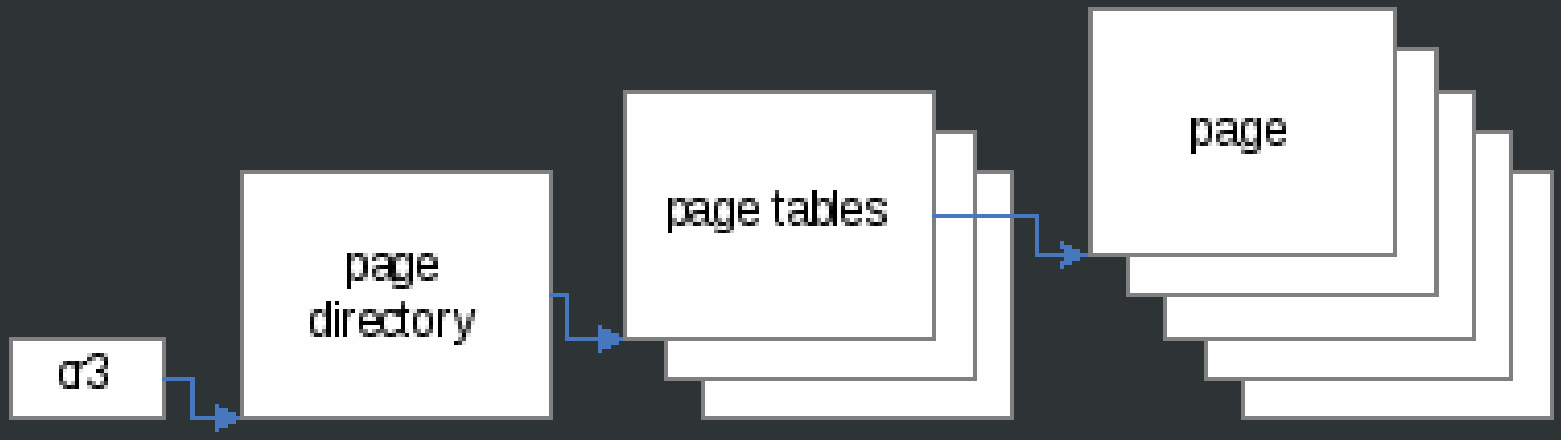
Useful Observations for Forensics

- All user processes have page tables
  - Note: it is possible for an advanced attacker to change the mode of the CPU so that paging is disabled
- Page directories and page tables can be identified using a set of known rules/patterns
- Other x86 data structures can also be identified with similar methods
- **Key point: page table identification can be used to locate hidden processes**

x86 Paging

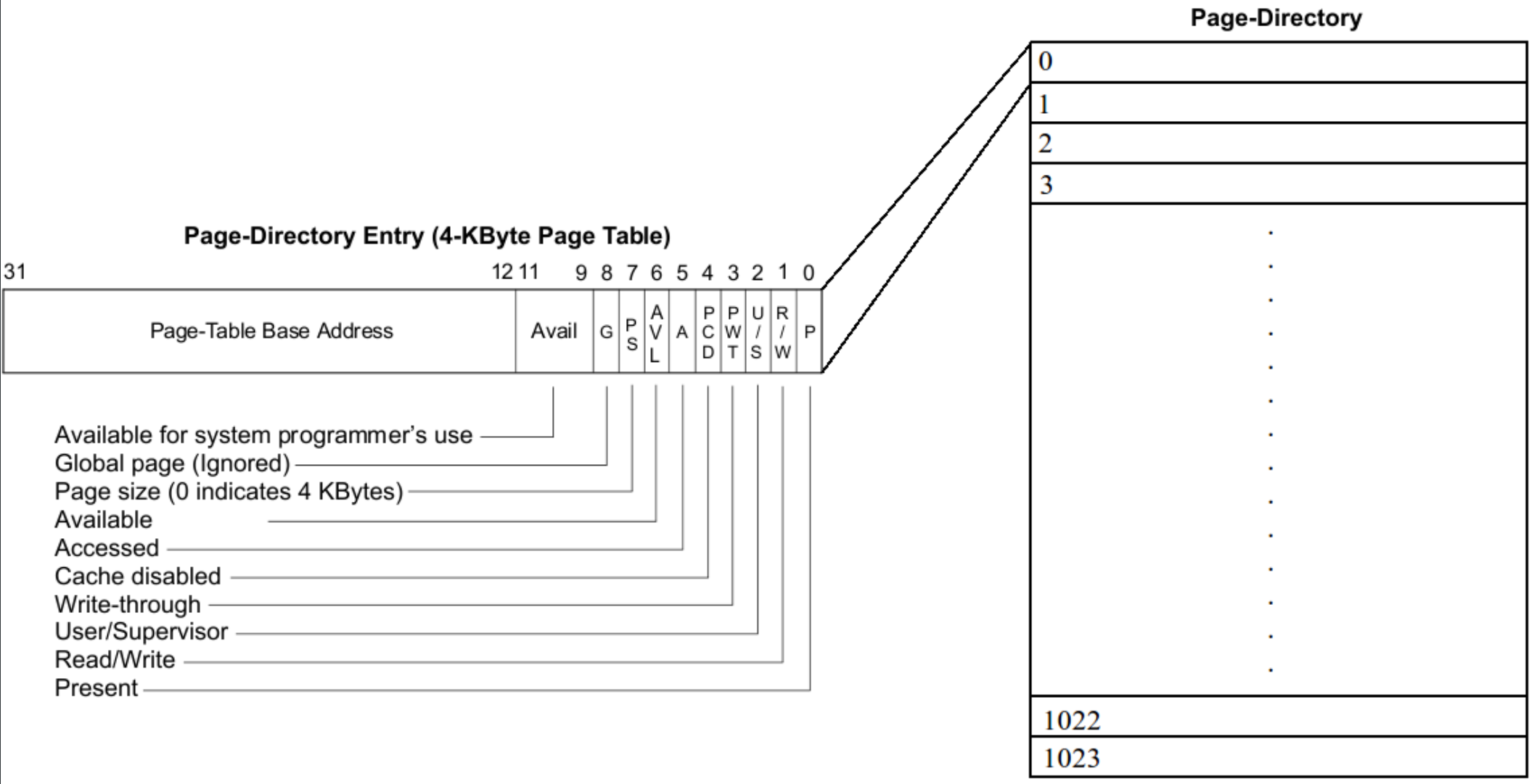
- x86 paging structures directly correlate with processes.
- Page tables isolate each process in a separate virtual address space
- Page tables contain a mapping from “virtual” addresses to physical addresses
- Paging structures have unique CR3 register values containing a physical address pointing to a page directory. This can be used to identify the process.

Page Table Overview



- CR3 Register: 32 bits
- Page Directory: 4KB.
  - Contains 1024-4 byte entries that can point to page tables or 4MB pages
- Page Table: 4KB. Contains 1024-4byte entries
- Page: 4KB of Data (or 4MB for large page)

In Depth: Page Directory



Entry Flags

- Plan: Detect Kernel Mappings in Page Directory
- All processes must map in the kernel to interact with kernel
- Traverse memory in 4K blocks looking for kernel mappings
- There will be roughly *1/4*(size of memory in MB)* mappings because of the 4MB pages

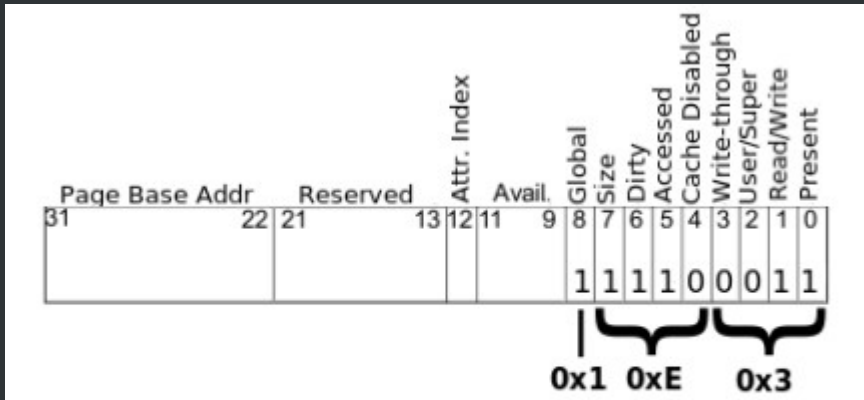


Figure 2: Page-Directory Entry Values for a 4MB Page (Linux Kernel Mapping)

Entry Flags & Results

Table 1: Algorithm Parameters for Different Operating Systems

Platform	Flag Values	Flag Threshold Count	Flag Location°
Linux (Non-PAE)	0x1E3	$\approx \max\left(\frac{1}{4} * (mem\_size\ MB), \frac{1}{4} * 896\right)$	pg_dir entries 768-1023
Linux (PAE)	0x1E3	$\approx \max\left(\frac{1}{2} * (mem\_size\ MB), \frac{1}{2} * 896\right)$	pg_dir 3
Windows (Non-PAE)	0x63	(350 - 500)	pg_dir entries 512-1023
Windows (PAE)	0x63	(350 - 500)	pg_dir 2 and 3

° *Zero Indexed (Page Directories Entries from 0-1023, PAE Page Directories from 0-3)*

Table 2: Example Test Results for Clean Memory Images

Distribution	Kernel Vers.	Size	Mode	CR3s	Missed	Expired
Centos 5.1	2.6.18-53.el5	512MB	Non-PAE	43	0	3
Fedora 7	2.6.21-1.3194.fc7	256MB	Non-PAE	93	0	6
Ubuntu Server 6.06.2	2.6.15-51-server	128MB	PAE	30	0	10
Ubuntu Server 8.04	2.6.24-19-server	128MB	PAE	25	0	7
Ubuntu 8.04	2.6.24-19-generic	512MB	Non-PAE	88	0	12
Windows XP SP1	N/A	512MB	Non-PAE	68	0	N/A
Windows XP SP2	N/A	256MB	PAE	61	0	N/A

Terminated Processes

- For forensics purposes, we can compare our reported process list with a list of expected processes, as reported by the operating system.
- Processes that have been legitimately terminated by the OS may still be resident in memory
  - Ex: In Linux systems, the OS will write a pointer in entry 0 of the page directory. (If the page directory is live, entry 0 will always be 0x00000000)
- Finding an unexpected process in memory that was not legitimately terminated is suspicious

Output

- Comparing our algorithm's findings with the kernel task structures helps locate suspicious processes
- Our algorithm also finds terminated processes that have been legitimately removed from the task structure list by the kernel
- Processes that are not in the task structure list and have not been deleted the kernel are flagged for further analysis.

```
...
Match: 0x0DA39000 - dd.    terminated = F
Match: 0x0DA15000 - sshd.  terminated = F
WARNING!: process at 0x0D9BB000; no match.
    # userspace entries: 3 terminated = F
Match: 0x0D959000 - sshd.  terminated = F
Match: 0x0D954000 - hald.  terminated = F
Match: 0x0D8AB000 - sshd.  terminated = F
Process at 0x0D888000; no match.
    # userspace entries: 1 terminated = T
Match: 0x0D87A000 - bash.  terminated = F
...
```

Background on Virtualization

- “Virtualize” all of the hardware resources of an x86-based computer, including a separate memory space
- Allows multiple operating systems to run on the same physical machine
- Used for cross-platform development, special security schemes, controlled environments
- Examples: VMWare, Xen, VirtualBox, Parallels

Application: Detect Virtualization

- In virtualization, a guest kernel will look different than the host kernel, allowing us to easily locate and separate all processes
- Processes for both host & guest can be analyzed simultaneously
- This could potentially be used to detect hypervisor rootkits

Table 3: Partial Output of Processing a Linux Memory Image with a Running Linux Virtual Machine (VMWare)

Physical Address*	Num. 0x1E3 Entries	Num. User Space Entries
0x1EDE1000	127	48
0x1E80F000	127	9
0x1D83B000	127	10
0x1D833000	127	3
0x1D5E2000	79	6
0x187D4000	79	15
0x10008000	79	8
0x05B08000	79	19

All Together:

```
197: page-directory-pointer table base address (CR3): 0x1747A000
       0:addr: 0x1F9FC000,  num_flags: 0     num_4k_ent: 8     num_2M_ent: 0   (userspace)
       1:addr: 0x1FE89000,  num_flags: 0     num_4k_ent: 25    num_2M_ent: 0   (userspace)
       2:addr: 0x0C752000,  num_flags: 480   num_4k_ent: 467   num_2M_ent: 13  Windows
       3:addr: 0x1F9F9000,  num_flags: 391   num_4k_ent: 391   num_2M_ent: 0   Windows
199: page-directory-pointer table base address (CR3): 0x146B6000
       0:addr: 0x1830D000,  num_flags: 0     num_4k_ent: 7     num_2M_ent: 0   (userspace)
       1:addr: 0x1818E000,  num_flags: 0     num_4k_ent: 20    num_2M_ent: 0   (userspace)
       2:addr: 0x1830F000,  num_flags: 480   num_4k_ent: 467   num_2M_ent: 13  Windows
       3:addr: 0x1830C000,  num_flags: 390   num_4k_ent: 390   num_2M_ent: 0   Windows
197: page-directory-pointer table base address (CR3): 0x1038B000
       0:addr: 0x18DDF000,  num_flags: 0     num_4k_ent: 8     num_2M_ent: 0   (userspace)
       1:addr: 0x18D60000,  num_flags: 0     num_4k_ent: 9     num_2M_ent: 0   (userspace)
       2:addr: 0x18E61000,  num_flags: 480   num_4k_ent: 467   num_2M_ent: 13  Windows
       3:addr: 0x18CDE000,  num_flags: 389   num_4k_ent: 389   num_2M_ent: 0   Windows
...
490  pg_dir_base_addr: 0x37153000,  num_flags: 224  num_userspace_ent: 14        Linux
491  pg_dir_base_addr: 0x37130000,  num_flags: 224  (terminated)                 Linux
492  pg_dir_base_addr: 0x3712C000,  num_flags: 224  num_userspace_ent: 9         Linux
493  pg_dir_base_addr: 0x37127000,  num_flags: 224  num_userspace_ent: 9         Linux
494  pg_dir_base_addr: 0x37122000,  num_flags: 224  num_userspace_ent: 4         Linux
495  pg_dir_base_addr: 0x370CA000,  num_flags: 224  num_userspace_ent: 162       Linux
496  pg_dir_base_addr: 0x370BD000,  num_flags: 224  (terminated)                 Linux
```

Figure 3: This partial output contains the analysis of a 1GB memory image with a Non-PAE Linux (Ubuntu 8.04) host containing a 512MB PAE Windows (XP SP2) guest virtual machine. The Windows XP entries show the page directory pointer table and four corresponding page directories. The Linux entries marked "terminated" have a pointer in entry 0 and no other userspace entries.

Assumptions & Limitations

- We assume no false page directories have been intentionally inserted that match our expected pattern.

- We assume paging is turned on. It is possible to create a process that runs with paging turned off, but this is much more difficult than using the existing support for process creation.

Current Status & Future Work

Current:

- Publication in Journal of Digital Investigation (Elsevier) in next issue
- Filed an IP Disclosure with JHU/APL Office of Technology Transfer

Future Work:

- Hypervisor Detection – Model common hypervisor's page tables (Xen, Virtual Box) and research hypervisor rootkit detection
- Expand Windows research to Vista/7
- Continue rootkit and malware detection as it evolves