The Research on Linux Memory Forensics

To cite this article: Jun Zhang and ShengBing Che 2018 IOP Conf. Ser.: Mater. Sci. Eng. 322 052021

View the article online for updates and enhancements.

You may also like

- Improving Real Time Performance of Linux System Using RT-Linux
  Callng Wang, Fan Yang, Hongwei Wang et al.

- Key Technologies of Phone Storage Forensics Based on ARM Architecture
  Jianghan Zhang and Shengbing Che

- A study on windows-based ransomware implications on linux operating system using compatibility layer wine based on dynamic analysis
  Rycka Septiasari and Yogha Restu Pramad
The Research on Linux Memory Forensics

Jun Zhang¹, ShengBing Che²

¹School of Central South University of Forestry and Technology, ChangSha, China
²School of Central South University of Forestry and Technology, ChangSha, China

*Corresponding author e-mail: 244767346@qq.com

Abstract. Memory forensics is a branch of computer forensics. It does not depend on the operating system API, and analyzes operating system information from binary memory data. Based on the 64-bit Linux operating system, it analyzes system process and thread information from physical memory data. Using ELF file debugging information and propose a method for locating kernel structure member variable, it can be applied to different versions of the Linux operating system. The experimental results show that the method can successfully obtain the system process information from physical memory data, and can be compatible with multiple versions of the Linux kernel.

1. Introduction
Memory forensics is a very important part of computer forensics, which is mainly for some volatile memory data. For example, it is from the computer's physical memory and exchange page files to find, analyze, extract the evidence. It can effectively combat with some high-tech computer crime. Memory forensics technology has been developed for many years, especially in 2005, DFRWS (Digital Forensic Research Workshop) [1] held a memory analysis challenge contest, after that, memory forensics attracts more and more researchers, and forensics tools are springing up like mushrooms, such as Expert Witness, TSK, volatility, SMART, and so on.

In the Windows operating system, the kernel is completely controlled by Microsoft. But in the Linux operating system, it is an open source kernel, so the kernel of some data structure member’s offset may have been changed, this will bring difficulties to memory forensics work which based on Linux.

In view of the above problems, this paper proposes a method to dynamically locate kernel data structure member offsets which is based on ubuntu linux operating system, and get the process information from the physical memory data through this method.

2. Dynamic locating kernel data structure member’s offset from ELF file’s debugging information
The operating system kernel contains many data structures that involve all aspects of operating system management, these data need to be analyzed in memory forensics, it is important to understand the information about these data structures.
2.1. The principle of dynamically getting the kernel data structure member’s offset

An ELF file with debugging information include the name of the structure, the type of member, the name of member, the offset of member, and the size of member. In Linux, the kernel-driven ko file is an ELF file too, if the debugging information is included, the ko file will contain some kernel structure information. We can use pahole tool to resolve debugging information. As shown in Figure 1, it is a structure information which is printed by pahole.

![Figure 1. Pahole example](image)

The results are regular, and we can use regular expressions to get information for the specified data structure. A structure may contain array or some other data type, so we might need multiple regular expressions.

2.2 Method evaluation

This dynamic locating method will be very effective if the kernel does not change significantly. If there is a big change in the Linux kernel, for example, a structure is deleted, this method will not correctly parse the details of the structure information. But, if the above problems occurred, the analysis method of memory forensics will be changed too.

3. Linear address translation in Linux Kernel

3.1. Locate the kernel PGD

In the Linux operating system, using the Page Global Directory (PGD) to translate linear addresses to physical addresses. The kernel variable init_mm.pgd holds the value of the kernel's PGD, and init_mm is a mm_struct structure. We can read the value of the kernel’s PGD by locating the position of init_mm, and then parse the offset of the pgd member.

The address of init_mm can be obtained from the kernel symbol export table. By default, the Linux system’s kernel symbol export table is in the /boot/System.map.xxx file, as shown in Figure 2, it is a part of the kernel symbol export table.

![Figure 2. kernel symbol export table example](image)

We can find the init_mm linear address from the kernel symbol export table. As shown in Figure 3, it is init_mm’s address information in the kernel symbol export table.

![Figure 3. init_mm’s address example](image)

Form the Figure 3, the address of init_mm is 0xffffffff81e6ffe0, it is a linear address, we need to get its physical address. The symbols listed in the kernel symbol export table are in the linear mapping area, there is a fixed offset between the virtual address and the physical address. We can find the address information by phys_startup_64 and startup_64 in the 64-bit kernel symbol export table. As shown in Figure 4, it is the address information.

![Figure 4. phys_startup_64 and startup_64](image)
The address of phys_startup_64 is physical address, and the address of startup_64 is linear address. Based on the offset of these two addresses, we can get the offset of physical address and linear address is 0xffffffff8000000. So, init_mm’s physical address is 0x1e700a0. According to the method which described in 2.1, we can get the pgd’s offset in mm_struct is 0x40 and pgd’s size is 8 bytes. As shown in Figure 5, it’s the physical memory data at 0x1e700e0.

![Physical Memory Data](image)

**Figure 5.** Physical memory data

From the physical memory data, we can know that the pgd’s value is 0xFFFFFFFF81E0A000, this address is a linear address, it minus 0xffffffff8000000 is the physical address 0x1e0a000, this address is kernel’s Page Global Directory.

### 3.2. Translate linear address to physical address

The modern Linux operating system is the protection mode, and the linear address used in the memory forensics needs to be translated to the physical address through the page table. Linux use a common four-level paging model that applies to both 32-bit and 64-bit operating system [2], this four-level is divided into Page Global Directory (PGD), Page Upper Directory (PUD), Page Middle Directory (PMD), and Page Table (PT). As shown in Figure 6, it’s the four-level paging model.

![Four-Level Paging Model](image)

**Figure 6.** Four-level paging model

In the Linux four-level paging model, the lower 48 bits of the linear address are valid bits for the paging partition. In the 4KB page, the four-level paging model is divided into 9: 9: 9: 12, 9 bits is the PGD index, 9 bits is the PUD index, 9 bits is the PMD index, 9 bits is the PT index, 12 bits is the page offset, and this is a small page mode. Large page mode is supported in Linux too, the four-level paging model will be divided into 9:9:9:21(2MB Page) and 9:9:30(1GB Page). The difference between large page mode and small page mode is only the size of the page at the last level. When reading each level address, we need to check whether it’s a large page model or a small page model, if it is a large page mode, all remaining address bits are computed as page offsets, otherwise we continue calculating address as 9:9:9:12 mode. The 7th bit of the linear address is used to distinguish whether it is a large page or small page mode, if the 7th bit of the linear address is set to 1, then the large page mode is represented, otherwise the small page mode is represented.

During address translation, 4KB page alignment is required. High 16-bit and low 12-bit of 64-bit linear addresses are attribute tags, every level linear address need ‘and’(&) operation with the address 0x0000FFFFFFFF000, the last level address is page address, and page address plus page offset is the final physical address.
For example, the following translation operation is the translation of the kernel linear address to the physical address in 4KB small page model.

Kernel’s virtual address is 0xffffffff801386586000 and kernel’s PGD Physical Address is 0x1e0a000. Divide the low 48 bits of the virtual address in 9:9:9:9:12 mode, the binary value of the virtual address is 1000100010 111000110 01011000 011000000000, the index information at all levels is PGD Index: 0x110, PUD Index: 0x4, PMD Index: 0x1C3, PT Index: 0x58, PAGE Offset: 0x600.

1. Calculate the PUD directory address by the PGD directory address and the PGD index value. 
   \[ \text{PGD} + 0 \times 8 \times 0 \times 110 = 0 \times 1E0A000 + 0 \times 880 = 0 \times 1E0A880 \]
   Read the memory data at physical address 0x1E0A880, as shown in Figure 7.

   ![Figure 7. memory data](image)

   The address is 0x0000000000220c067, translate it to binary is 10001000001100000000 000000000000, bit 7 is marked as 0, which means that this is a small page mode, remove the high 16 and low 12 bit attribute value, the result is 0x220C000, this value is the PUD address.

2. Calculate the PMD directory address by the PUD directory address and PUD index value.
   \[ \text{PUD} + 0 \times 8 \times 0 \times 0 \times 4 = 0 \times 220C000 + 0 \times 20 = 0 \times 220C020 \]
   Read the memory data at physical address 0x220C020, as shown in Figure 8.

   ![Figure 8. memory data](image)

   The address is 0x0000000000220F067, translate it to binary is 10001000001111000000 000000000000, bit 7 is marked as 0, which means that this is a small page mode, remove the high 16 and low 12 bit attribute value, the result is 0x220C000, this value is the PMD address.

3. Calculate the PT page table address by the PMD directory address and PMD index value.
   \[ \text{PMD} + 0 \times 8 \times 0 \times 1 \times C3 = 0 \times 220F000 + 0 \times 0E18 = 0 \times 220FE18 \]
   Read the memory data at physical address 0x220FE18, as shown in Figure 9.

   ![Figure 9. memory data](image)

   The address is 0x00000001386D7063, translate it to binary is 10011100001101101101100000 000000000000, bit 7 is marked as 0, which means that this is a small page mode, remove the high 16 and low 12 bit attribute value, the result is 0x1386D7000, this value is the PT page table address.

4. Calculate the page table address by the PT page table address and PT index value.
   \[ \text{PT} + 0 \times 8 \times 0 \times 58 = 0 \times 1386D7000 + 0 \times 2C0 = 0 \times 1386D72C0 \]
   Read the memory data at physical address 0x1386D72C0, as shown in Figure 10.

   ![Figure 10. memory data](image)

   The address is 0x800000138658163, remove the high 16 and low 12 bit attribute value, the result is 0x1386580000, this value is the page address.

5. Calculate the final physical address by the page address and page offset.
   \[ \text{PAGE} + \text{OFFSET} = 0 \times 1386580000 + 0 \times 600 = 0 \times 138658600 \]
0x138658600 is the final physical address of liner address 0xFFFF880138658600.

The method above is 9:9:9: 9:12 small page model, the 9:9:9:21 and 9:9:30 large page model are similar to the above.

4. Enumerate processes in the linux kernel
Linux kernel use task_struct structure to describe a process. There are many member variables in the task_struct structure, as shown in figure 11, it is part of the field in task_struct.

```c
struct task_struct {
    volatile long int state; /* 0 8 */
    struct list_head tasks; /* 848 16 */
    struct mm_struct * mm; /* 928 8 */
    struct mm_struct * active_mm; /* 936 8 */
    pid_t pid; /* 1096 4 */
    char comm[16]; /* 1536 16 */
};
```

**Figure 11.** task_struct information

The state field describes the state of the current process. The tasks field is a two-way circular linked list structure. The mm field is the memory descriptor for the current process, if mm is NULL, then the task_struct is a thread. The active_mm field is the memory descriptor used by the current task_struct. The pid field is the process id. The comm field is the name of the process.

Linux process two-way circular linked list’s header is init_task, init_task can be found in the kernel symbol export table. As shown in figure 12, it’s a two-way circular linked list of kernel task_struct.

**Figure 12.** two-way list of task_struct

We can get all the process descriptor information from two-way circular linked list of task_struct.tasks.

5. Conclusion
Based on Linux, this paper focuses on the method of memory address translation, process enumeration in memory forensics. Based on the debugging information of ELF file, a common method for locating the member variables of kernel structure is proposed, in this method, the member variable’s offset information in a process descriptor (task_struct) is dynamically acquired. Linear address to physical address and enumeration of kernel process have been achieved.
References
[1] DFRWS. DFRWS 2005 forensics challenge[EB/OL]. (2005-08)[2012-05-01].http : //www.dfrws.org/2005/challenge/index.shtml.
[2] Daniel P. Bovet, Marco Cesati. Understanding the Linux Kernel. O'Reilly Media,2007.
[3] Emmerich W. Distributed Component Technologies and their Software Engineering Implications[M]. ACM Press,2002.
[4] Ming Xin Zhao.Trim and Transplantation of Linux Kernel Based on ARM[J]. Advanced Materials Research,2013, (56):72-76.
[5] Richard Stevens, Stephen ARago. Advances Programming in the UNIX Environment[M]. American: Addison Wesley/Pearson,2006.