Investigating the PROCESS block for memory analysis

Khairul Akram Zainol Ariffin
Universiti Kebangsaan Malaysia
20 PUBLICATIONS **75** CITATIONS
SEE PROFILE

Ahmad Kamil Mahmood
Universiti Teknologi PETRONAS
175 PUBLICATIONS **776** CITATIONS
SEE PROFILE

Jafreezal Jaafar
Universiti Teknologi PETRONAS
171 PUBLICATIONS **776** CITATIONS
SEE PROFILE

Investigating the PROCESS block for Memory Analysis

KHAIRUL AKRAM ZAINOL ARIFFIN
Computer Information Science
Universiti Teknologi PETRONAS
Perak, MALAYSIA
akram.zainolariffin@gmail.com

AHMAD KAMIL MAHMOOD
Computer Information Science
Universiti Teknologi PETRONAS
Perak, MALAYSIA
Kamilmh@petronas.com.my

JAFREEZAL JAAFAR
Computer Information Science
Universiti Teknologi PETRONAS
Perak, MALAYSIA
jafreez@petronas.com.my

Abstract: - Over the past few years, memory analysis has been an issue that has been discussed in digital forensics. A number of tools have been released that focus on memory acquisition of Windows system. However, the implementation of memory analysis is still limited as it encounters a lot of difficulties. The aim of this paper is to outline one of the difficulties with regards to the structure of EPROCESS block. It will discuss about the differences in offset between Windows 2000 and Window XP. Further, the important of internal structures in EPROCESS block will be identified as they play an important role in the analysis and theory reconstruction for forensic investigation. Nevertheless, an address translation for x86 platforms will be demonstrated in this paper. Hence, the limitation of the address translation algorithm will also been discussed and identified.

Key-Words: - Operating System, Digital Forensic, Memory Analysis, Algorithm Design

1 Introduction

As the computer becomes common in our daily life, the number of cybercrime is also increasing over the years. The Council of Europe's Cybercrime Treaty defines cybercrime as a range of crime that is committed using computer, network and hardware. In general, Symantec divides the cybercrime into two categories:[1]

- Type 1: a single event from victim's perspective which is facilitated by the crimeware such as keystroke logger, viruses, rootkit and Trojan.
- Type 2: refers to the activity such as cyber stalking, harassment, blackmail, complex corporate espionage and stock market manipulation.

In order to overcome this crime, a field known as Digital Forensic has been established. It is referred as a proven scientific approach in preserving, collecting, validating, identifying, analyzing, documenting and presenting the contents of computer as evidence for cybercrime[2, 3]. Recently, the research in Digital Forensic has moved its attention to include volatile analysis in investigation. It started in 2005 where Digital Forensic Research Workshop (DFRWS) organized a Windows Memory Challenge [4]. During the event, a tool known as Memparser [5] which was developed by Chris Betz has been introduced. This tool allows the investigators to load the memory dump of the Windows systems, reconstruct the process information and extract the data from the process. In the same year, KntList [6] has been developed by Garner Jr which has a capability to acquire and examine the memory dump from a live Windows system.

Although, the importance of memory analysis has been announced during the event, most investigators normally ignore this approach and only rely on the traditional procedure. In the traditional procedure, the evidence is collected only from the non volatile device with the help of powerful software such as EnCase. Burdach, a well known forensic researcher also listed out the factors that contribute to this scenario during his presentation in February 2006**:**

- The traditional approach is simple and easy.
- Modification of the exhibit is almost impossible in traditional approach if it is handled with care.
- The traditional method is well-known to most investigators as it has already been established for a long period of time.

However, relying on non- volatile analysis alone will not establish a good conclusion in justifying the crime scenario. Further, anti forensic tool such as encryption is available as freeware which can counter the investigation process. The sensitive information such as password, encryption keys and username is not available in non-volatile device but in volatile memory as it is used to store temporary processes and data before they are transmitted to Central Processing Unit (CPU) for operation.[7]

2 Theory

2.1 Computer System Architectures and Windows

The system architecture is affected by the component or hardware that has been installed in the computer system. The type of processor that has been installed in the system will affect the operation of the Operating System. In general, Windows operating system is designed to be able to run in a variety of hardware architecture. Depending on the processor, the system architecture for Windows can be divided into two categories such as 32-bit and 64-bit systems. Table 1 summarizes the existing platform for both 32 bit and 64 bit system.

Table 1 Hardware platform for 32 and 64 bit system

System	**Hardware Platform**
32 bit	• x86 system with non-PAE (Physical Address Extension) • x86 system with PAE
64 bit extended	• IA64 system • x64 system

Although the processes or mechanisms in operating system remain the same, the algorithm that applies to each mechanism will be different between the systems. Even within the same platform, the differences in algorithms will still exist. As an example, the system that has been installed with Intel x86 Pentium Pro Processor has different characteristic with the previous version. In fact, this processor has introduced Physical Address Extension (PAE) which has an exactly different address translation algorithm compared to the previous platform. This scenario has been demonstrated in Russinovich and Solomon [8] where each platform has different algorithms for address translation in the memory. Further, it also explains that the system architecture will also affect the page size, the maximum amount of storage available, and address representation.

Apart from that, the Windows architecture also plays an important role in understanding the activities of the operating system. In theory, Windows architecture is defined by two mode such as user mode and kernel mode. In user mode, there are four main components such as system support processes, service processes, user processes application and environment subsystem with each of them has their own private process address space. Thus, the internal structure such as threads will execute in the protected process address space. On the other hand, the kernel component consists of five important components such as Windows executive, windows kernel, device driver, hardware abstraction layer (HAL) and windowing and graphic system. Window executive and Windows kernel will play their role in the operating system management and operation (For example, the function in Windows kernel is responsible for threads scheduling of the system). On the other hand, device drive and hardware abstraction layer (HAL) will manage the device or hardware that is responsible in the task or operation. Finally, the windowing and graphic system will implement the graphical user interface (GUI) function which will deal with Windows, user interface controls and drawing.

2.2 Object and Important Internal Structure in the Memory

The theories in object and internal structure have been outlined in both Dhananjay, M. [9] and Russinovich and Solomon [10]. Both explain about the function of internal structure with Dhananjay, M. focussing on general operating system while Russinovich and Solomon only concentrate towards Windows operating system. In general, an object is an important entity that the kernel or central operating system uses in order to keep the system runs. Windows divide the objects into three categories such as executive, kernel, and GDI/User objects. The executive object is created by either environment subsystem on behalf of the user application or by various component of operating system. In case of creating a file, a kernel function (CreateFile) will be called to validate and initialize the data before an executive file object is created. The examples of executive objects are Process, Thread, Job, Section, File and so on. Nevertheless, each of the objects has the object header and object body. The object header will point towards the type object which contains the information that is common to each representative of the object. This object header is controlled by the object manager during the task performing as the systems runs (i.e. process manager controls the process object by providing a common and uniform mechanism). Figure 2 shows the representation of object structure.

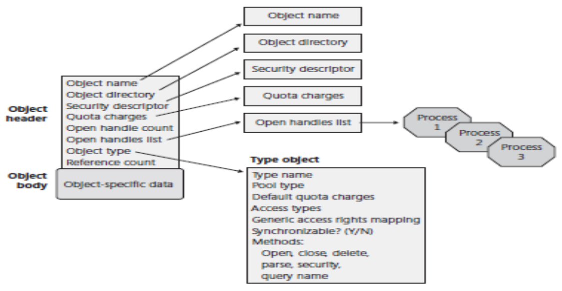


Fig 2: Object structure [10]

Amari, K. [11] demonstrates the way the kernel operates to store the data in the object. In the demonstration, it illustrates that the kernel has two sets of memory storage which is known as pool to store the object. The set of pool is divided into paged pool and non-paged pool. The way that the kernel stores the object is based on its importance for the system to run. Most data are stored in paged pool as they can be placed as a file in the hard disk if the machine is running low on the physical memory [12]. However, the non-paged pool will store the important objects that the kernel needs to access frequently. Examples of important objects are the processes and threads. Due to this fact, it means that the running process at the time of the physical memory being captured will be available to the forensic analyst [13].

In the volatile memory storage, one of the important structures that store a critical data with regards to the object is known as a handle where it stores the address of the linking object. As an example, when a process creates or opens a structure by name, it receives handle which represents its access to the structure. Thus, for all user mode processes, they must own a handle to a structure before their threads can be used by it. Hence, the handle serves as an indirect pointers to system resources for operation in operating system.

Apart from that, memory manager also maintains a set of Virtual Address Descriptors (VADs) for each process with the purpose of tracking the status of process's address space. This VAD will store the information on the attributes of the object (i.e. range of address, private/share, inheritance of child, security). Finally, a Section Object which is also known as file mapping object is a block of memory where a number of processes can share. Due to the theory of VAD, a tool known as VADtool has been developed which is useful in tracking the memory mapped files from memory dump [14].

2.3 Address Translation for Windows System

The address translation for a running Windows system has been demonstrated in [10]. In general, the algorithm for address translation in Windows system is affected by the system architectures. Each of the architectures has its own address translation algorithms. It is due to the fact that each of the architectures uses different amount of page table in order to translate the virtual address of an internal structure into the physical address. As an example, Windows in x 86 hardware platforms without PAE will only use TWO LEVEL page tables to translate the virtual address into physical address. Nevertheless, the differences in the algorithm also affected the representation of the address of the object.

2.4 Memory Acquisition

Although volatile memory provides important information with regard to investigation, there is also a setback when using volatile approach. It is due to the fact that the contents within memory will be lost once the system is shut down. Therefore, as a first step, the investigator needs to question the suitability, availability, and reliability of the volatile content that reflect the investigation. Plus, the acquisition of volatile memory plays a serious part in the investigation. The acquisition of volatile memory can be divided into two: [15],[16]

1) Software Based
    - It uses a trusted toolkit to capture the memory dump. Thus, every action is performed on the system.
    - Example: GNU dd, kntDD
    - Acquisition is done by applying a code: dd.exe if = \\.\PhysicalMemoryof = \\<remoteshare>\memorydump.exe
    - Can also be obtained through crash dump.
2) Hardware Based
    - It is done without relying on Operating System. The CPU is suspended and Direct Memory Access (DMA) is used to copy the contents.
    - Tools: TRIBBLE or FIREWIRE [5]

2.5 Persistence of Data that is stored in the memory

According to the work that has been done by Garfinkel, Chow and Rosenblum in 2004, it can be concluded that 86% of the contents in the memory has not changed their value and location [17]. Thus, the metadata about the processes and other objects can still survive in the memory when the machine is still in use. Their work has been divided into two categories where the first test is to find the effect on data when the machine has been used up to 28 days. The second test is to justify the content of the memory when the machine has been rebooted. The summary of the test one and two are shown in table 2 and 3 respectively.

Table 2 Effect on memory's contents with respect to the duration of the computer being used

Duration	Description
14 days	23Kb of data still remains in the memory
28 days	7Kb of data remains in the memory. This size of data is enough to contain the information on password and cryptographic keys.

Table 3 Effect on memory's contents with respect of the reboot undertaken on the machine

Reboot	Description
Soft reboot	It leaves most of the data still intact in the Random Access Memory (RAM)
Hard reboot	The data will remain up to 30 seconds without power

2.6 Memory Analysis Techniques

2.6.1Strings Search

String search is a process of finding of anything that is recognizable across the memory dump. The strings can either be in UNICODE or ASCII formats. The reason behind this technique is because some letter in memory dump is likely to occur quite frequently. XORSearch [18] is a tool that is designed based on this technique. It takes a keyword as an input and then performs the search throughout the memory dump. The user is allowed to specify the length of the string of character. Further, the tool can also help to find the keyword that has obfuscated by using either Exclusive OR (XOR) or Rotate Left (ROL) function that comes with it.

2.6.2 Enumerating the Running Processes

Windows OS represents each process in volatile memory by EPROCESS block. It is a block that contains pointers to both next and previous EPROCESS blocks. It can be illustrated as in Figure 3.

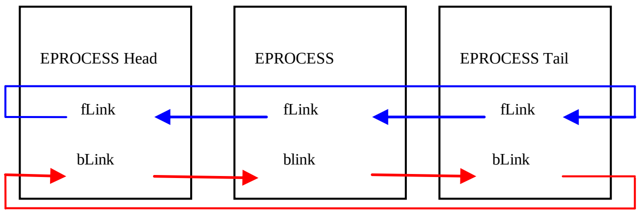


Fig 3 The doubly linked list chain of EPROCESS in normal operation with backward (bLink) and forward (fLink) links [19], [20]

EPROCESS block includes the information on attributes of the process along with pointer to other data structures that is related to it. One of these structures is Process Environment (PEB) which locates the location of executable files and the DLL's path. Due to this fact, AccessData [16] group has designed a tool that is known as Forensic Toolkit (FTK). This tool will parse the EPROCESS block to enumerate all the contents within memory. It also makes use of Directory Table Base (DTB) to create physical to virtual mapping in order to identify the process in memory. Further, Windows Memory Forensic Toolkit (WMFT) has applied this technique by tracking the PsActiveProcessHead link to capture all the active processes in the memory dump [21]. Apart from that, Ruichao Zhang, Lianhai Wang, Shuhui Zhang [22] has demonstrated the data extraction from memory dump by using Kernel Processor Control Region (KPCR). In the demonstration, the information such as running processes, current network connection, file content and other data can be extracted from the image.

2.6.3 File Signature Search

In theory different types of files have different signatures. A signature is a specific pattern of value that is unique to a particular type of file. Thus, it allows the analysis to be done by using file carving. S. M. Hejazi, C. Talhi, M. Debbabi [23] has outline the use of aplication or protocol fingerprint to trace the active application in the memory. The test was conducted to track online application such as email and messenger where from the result, it showed that each of the application had a use fingerprint representation.PTFinder [24] is a tool that applies the file carving where the technique is done linearly to recover only the contiguous file. This technique is applicable because most operating systems will convert the file to be contiguous file instead of fragment files. [4]

3.0 Method

A combination of string search and basic mathematical operation is used to track the internal structure of the process block. The string search technique is applied to find the location of a known process. In general, the ImageFileName is the best choice to find the known process. Once the ImageFileName has been identified, the other internal structure such as threads link, SectionObject, process environment block (PEB), and VADRoot can be accessed by using the offset of the process structure. The basic mathematical algorithm such as addition and subtraction can be applied to allocate the position of the internal structure. Note that, the information that is stored in the process block can either be a data or the virtual address. If it is a data, it can be directly converted to useful information. However, if the information that is stored in the internal structures represents the virtual address, an address translation algorithm is needed to obtain the physical address. By doing so, the object that is attached to process block can be retrieved.

4.0 Experiment

4.1 Observation of EPROCESS block for Windows 2000 and Windows XP

This test is done while the computer is still running. The information about the offset of the EPROCESS block can be retrieved using Kernel Debugger.

4.2 Observation of Internal Structures in Process Block (Windows 2000)

In this test, the author will use the existing memory dump that is available in Digital Forensic Research Conference (DFWRS) website [4] . Since there are many researches that used this memory dump, therefore the author can directly compare the result with the work in the past. One of the works has been done by Andreas Schuster [25] using Volatility tool to retrieve UMG32.exe process.

The DFRWS memory dump is run using WinHex and string search techniques is applied to go to UMGR32.exe process block. Once the process block has been allocated, other information can be tracked directly or by applying an address translation algorithm. The algorithms to retrieve processID and DirectoryBaseTable are shown in figure 4 and 5 respectively.

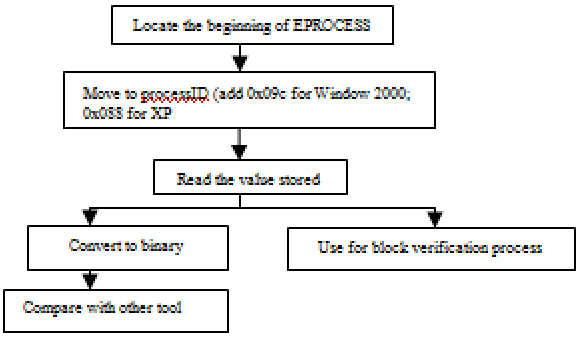

Locate the beginning of EPROCESS
Move to processID (add 0x09c for Window 2000; 0x088 for XP
Read the value stored
Convert to binary
Use for block verification process
Compare with other tool

Fig 4 Algorithm for ProcesID

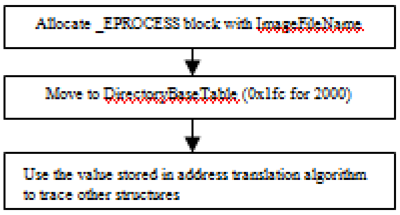

Allocate _EPROCESS block with ImageFileName
Move to DirectoryBaseTable (0x1fc for 2000)
Use the value stored in address translation algorithm to trace other structures

Fig 5 Algorithm to allocate DirectoryBaseTable in EPROCESS

4.3 Tracking the PEB block of UMGR32.exe using address translation.

The algorithm for address translation in memory dump is illustrated in figure 6. When applying this algorithm, it is assumed that the memory dump comes from a system with x 86 hardware platforms.

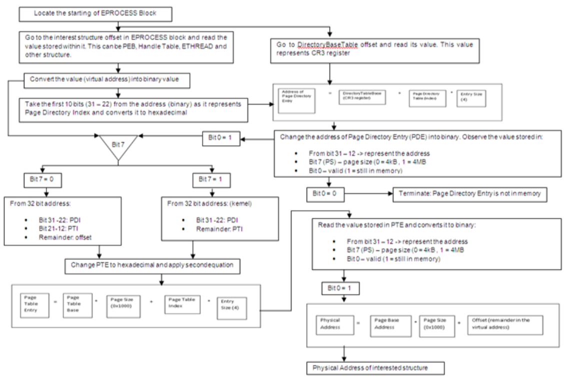
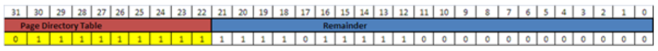

Locate the starting of EPROCESS Block
Go to the interest structure offset in EPROCESS block and read the value stored within it. This can be PEB, Handle Table, ETHREAD and other structure.
Go to DirectoryBaseTable offset and read its value. This value represents CR3 register
Convert the value (virtual address) into binary value
Take the first 10 bits (31 – 22) from the address (binary) as it represents Page Directory Index and converts it to hexadecimal
Address of Page Directory Entry
=
DirectoryTableBase (CR3 register)
+
Page Directory Table (Index)
*
Entry Size (4)
Bit 7
Bit 0 = 1
Change the address of Page Directory Entry (PDE) into binary. Observe the value stored in:
From bit 31 – 12 -> represent the address
Bit 7 (PS) – page size (0 = 4kB, 1 = 4MB
Bit 0 – valid (1 = still in memory)
Bit 7 = 0
Bit 7 = 1
Bit 0 = 0
Terminate: Page Directory Entry is not in memory
From 32 bit address:
Bit 31 -22: PDI
Bit 21-12: PTI
Remainder: offset
From 32 bit address: (kernel)
Bit 31 -22: PDI
Remainder: PTI
Read the value stored in PTE and converts it to binary:
From bit 31 – 12 -> represent the address
Bit 7 (PS) – page size (0 = 4kB, 1 = 4MB
Bit 0 – valid (1 = still in memory)
Change PTE to hexadecimal and apply second equation
Page Table Entry
=
Page Table Base
*
Page Size (0x1000)
+
Page Table Index
*
Entry Size (4)
Bit 0 = 1
Physical Address
=
Page Base Address
*
Page Size (0x1000)
+
Offset (remainder in the virtual address)
Physical Address of interested structure

Fig 6 Address translation algorithm

Step 1: Retrieve the virtual address of PEB

After obtaining the value that is stored in PEB structure, then converts it to a binary value. Since the memory dump comes from 32-bit system, the address translation mechanism uses two page system approach. The first 10 bits (from bit 31 to 22) is referred as the address of Page Directory Table. The remainder bits will represent either Page Table Index with address offset or Page Table Index alone. This characteristic depends on the size of Page Table. From the theory in memory management, the address of Page Directory Entry can be obtained by summing the CR3 register (DirectoryTableBase) with the Page Directory Table. However, the Page Directory table must be multiplied by 4 as each entry has a size of 4 bytes (e.g 0x075a700 + 0x1ff*4 = 0x075a77fc). Figure 7, 8, and 9 show the address of PEB in binary, the formula to obtain the Page Directory Entry and a WinHex presentation for Page Directory Entry respectively.



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Page Directory Table										Remainder																					
0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0

Fig 7: Address of PEB in binary form. The highlighted value represents the address for page directory table.

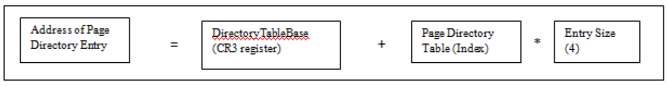
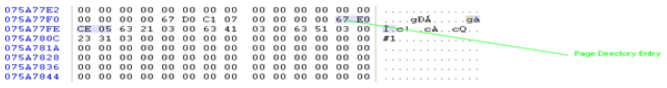

Address of Page Directory Entry
=
DirectoryTableBase (CR3 register)
+
Page Directory Table (Index)
*
Entry Size (4)

Fig 8: Formula to obtain the address of Page Directory Entry

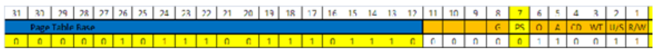


```
075A77E2  00 00 00 00 00 00 00 00  00 00 00 00 00 00  ..............
075A77F0  00 00 00 00 67 D0 C1 07  00 00 00 00 67 E0  ....gDÁ.....gà
075A77FE  CE 05 63 21 03 00 63 41  03 00 63 51 03 00  Î.c!..cA..cQ..
075A780C  23 31 03 00 00 00 00 00  00 00 00 00 00 00  #1............
075A781A  00 00 00 00 00 00 00 00  00 00 00 00 00 00  ..............
075A7828  00 00 00 00 00 00 00 00  00 00 00 00 00 00  ..............
075A7836  00 00 00 00 00 00 00 00  00 00 00 00 00 00  ..............
075A7844  00 00 00 00 00 00 00 00  00 00 00 00 00 00  ..............
```

Fig 9: value for Page Directory Entry. The value is read as 0x05cee067

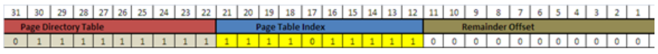
Step 2: Obtain the Page Table Base for PEB

The value that is retrieved from part (2) is then converted into binary as shown in figure 10 and the three regions that have been discussed in previous chapter are observed. Since the value in bit 7 is zero, therefore it is concluded that it has a small page size which equals to 4kB. As for bit 0, the value that is stored within it is ONE which means that the page table entry is still residing in physical memory. Due to this result, we can divide the virtual address of PEB in figure 7 into three regions as shown in figure 11. The address will be led by Page Directory Table, then followed by Page Table Index and finally the remainder represents the offset for the physical address.



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
Page Table Base																							G	PS	O	A	CD	WT	U/S	R/W
0	0	0	0	0	1	0	1	1	1	0	0	1	1	1	0	1	1	1	0	0	0	0	0	0	1	1	0	0	1	1

Fig 10: Page Table entry base for PEB



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
Page Directory Table										Page Table Index										Remainder Offset										
0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0

Fig 11: New Address section of PEB in binary

Step 3: Obtain the Page Table Entry from Page Table Index and Page table Base

Firstly, convert the binary value for Page Table Index and page Table Base into hexadecimal and perform the mathematical calculation by using the formula in Figure 12. Then, the value stored in the Page Table Entry is read. This value is referred to as the address of the page base. Figure 13 shows the value that is stored in the Page Table Entry.

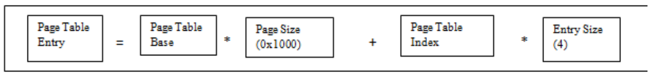
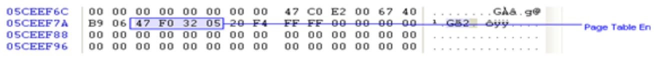


Fig 12: Formula to obtain the address of Page Table Entry.



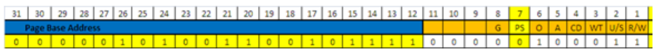
```
05CEEF6C  00 00 00 00 00 00 00 00  47 C0 E2 00 67 40  ........GÀâ.g@
05CEEF7A  B9 06 47 F0 32 05 20 F4  FF FF 00 00 00 00  ¹.Gð2. ôÿÿ....
05CEEF88  00 00 00 00 00 00 00 00  00 00 00 00 00 00  ..............
05CEEF96  00 00 00 00 00 00 00 00  00 00 00 00 00 00  ..............
```

Fig 13: Value stored in Page Table Entry.

Step 4: Obtain the physical address of PEB of UMGR32.Exe

The address of the page base is then converted to the binary and repeats the same step in (3) (e.g. observed the three regions on the address). From figure 14, it is shown that the page base address is valid and has a small size. Hence, we can obtain the physical address of the PEB by adding the page base address (in hexadecimal) to the offset of the physical address as shown in figure 15 (e.g. the remainder of the virtual address for PEB)

[0x0532f*0x1000 + 0 (offset) = 0x532f000]



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
Page Base Address																							G	PS	O	A	CD	WT	U/S	R/W
0	0	0	0	0	1	0	1	0	0	1	1	0	0	1	0	1	1	1	1	0	0	0	0	0	1	0	0	0	1	1

Fig 14: Page Base address

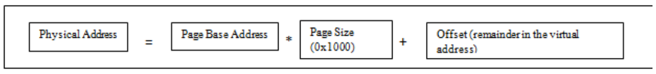
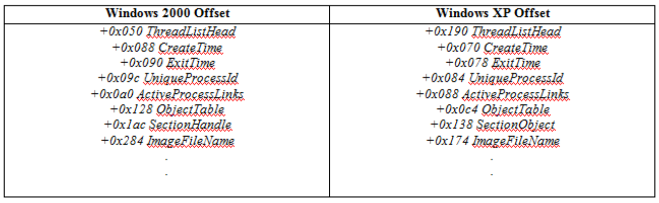


Fig 15: Formula to obtain the physical address for PEB of UMGR32.exe

5.0 Result and Discussion

From the first test, it can be concluded that although the internal structure between both operating system are the same, the offset of the structure is a bit different. The differences between the offset of these operating system are summarized in table 4.

Table 4: the differences between the offset of internal structures in Windows 2000 and Windows XP.

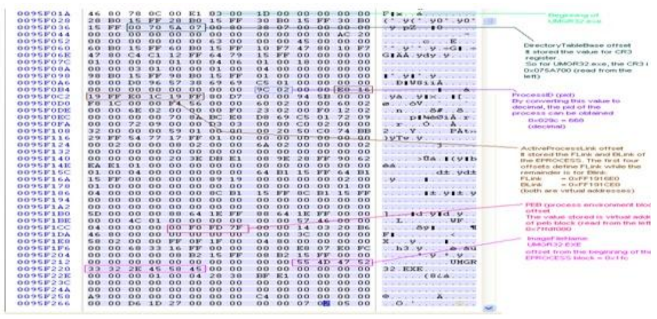


Windows 2000 Offset	Windows XP Offset
+0x050 *ThreadListHead*	+0x190 *ThreadListHead*
+0x088 *CreateTime*	+0x070 *CreateTime*
+0x090 *ExitTime*	+0x078 *ExitTime*
+0x09c *UniqueProcessId*	+0x084 *UniqueProcessId*
+0x0a0 *ActiveProcessLinks*	+0x088 *ActiveProcessLinks*
+0x128 *ObjectTable*	+0x0c4 *ObjectTable*
+0x1ac *SectionHandle*	+0x138 *SectionObject*
+0x284 *ImageFileName*	+0x174 *ImageFileName*
.	.
.	.

From table 4, although the offset representation remains the same, the value of the location is different. Thus, for the tools to work on different version of windows, some adjustments on the software coding need to be made. The other solution is by using known internal structures and calculates the different in range between them. The value that represents the difference of both structures will be unique for each Windows version. Hence it can be used as a selector to choose the type of Windows version. One of the ways is by using both ImageFileName and UniqueProcessID.

Unique value = ImageFileName offset – UniqueProcessID (1)

The offset gives an idea on where the critical internal structures are stored in the process block. When procede with the second test, the offset is used to obtain the critical structure and then translates the valued stored into the process information. As an example, after using the string search to allocate UMGR32.exe ImageFileName, the address of it needs to be deducted with the offset of ImageFileName (i.e. +0x284 for Windows 2000) to get to the beginning of the UMGR32.exe process block. Then, in order to move to other critical structure, the offset is added to the address (beginning of UMGR32.exe process block). Figure 16 represents the location of UMGR32.exe process block in memory dump and the result obtained from translating the internal structure of the block respectively.



```
0095F01A  46 80 78 0C 00 E1 03 00  1D 00 00 00 00 00
0095F028  28 B0 15 FF 28 B0 15 FF  30 B0 15 FF 30 B0
0095F036  15 FF 00 70 5A 07 00 80  38 07 00 00 00 00
0095F044  00 00 00 00 00 00 00 00  00 00 00 00 AC 20
0095F052  00 00 00 00 00 00 63 00  00 00 45 00 00 00
0095F060  60 B0 15 FF 60 B0 15 FF  10 F7 47 80 10 F7
0095F06E  47 80 C4 C1 12 FF 64 79  15 FF 00 00 00 00
0095F07C  01 00 00 00 01 00 04 06  01 00 18 00 00 00
0095F08A  00 00 03 01 00 00 01 00  04 00 00 00 00 00
0095F098  98 B0 15 FF 98 B0 15 FF  01 00 00 00 00 00
0095F0A6  00 00 D0 96 57 38 69 69  C5 01 00 00 00 00
0095F0B4  00 00 00 00 00 00 00 00  9C 02 00 00 E0 16
0095F0C2  19 FF E0 1C 19 FF 80 D7  00 00 94 5B 00 00
0095F0D0  F8 1C 00 00 F4 56 00 00  60 02 00 00 60 02
0095F0DE  00 00 6E 02 00 00 00 F0  23 02 00 F0 12 02
0095F0EC  00 00 00 00 70 8A BC E8  D8 69 C5 01 72 09
0095F0FA  00 00 72 09 00 00 D3 03  00 00 C0 02 00 00
0095F108  32 00 00 00 59 01 00 00  00 20 50 C0 74 BB
0095F116  29 FF 54 77 17 FF 01 00  00 00 00 00 00 00
0095F124  00 02 00 00 08 02 00 00  6A 02 00 00 00 02
0095F132  00 00 00 00 00 00 00 00  00 00 00 00 00 00
0095F140  00 00 00 00 20 3E DB E1  08 9E 28 FF 90 62
0095F14E  EA E1 01 00 00 00 00 00  00 00 00 00 00 00
0095F15C  01 00 04 00 00 00 00 00  64 B1 15 FF 64 B1
0095F16A  15 FF 00 00 00 00 89 19  00 00 00 00 02 00
0095F178  01 00 00 00 00 00 00 00  00 00 00 00 01 00
0095F186  04 00 00 00 00 00 8C B1  15 FF 8C B1 15 FF
0095F194  00 00 00 00 00 00 00 00  00 00 00 00 00 00
0095F1A2  00 00 00 00 00 00 00 00  00 00 00 00 00 00
0095F1B0  5D 00 00 00 88 64 1E FF  88 64 1E FF 00 00
0095F1BE  00 00 4C 01 00 00 00 00  00 00 57 46 00 00
0095F1CC  04 00 00 00 00 F0 FD 7F  00 00 14 03 20 B6
0095F1DA  46 80 00 00 00 00 00 00  00 00 3C 00 00 00
0095F1E8  58 02 00 00 FF 0F 1F 00  04 80 00 00 00 00
0095F1F6  00 00 68 33 16 FF 00 00  00 00 E8 07 E0 FC
0095F204  00 00 00 00 08 B2 15 FF  08 B2 15 FF 00 00
0095F212  00 00 00 00 00 00 00 00  00 00 55 4D 47 52
0095F220  33 32 2E 45 58 45 00 00  00 00 00 00 00 00
0095F22E  00 00 00 01 00 04 28 38  BF E1 00 00 00 00
0095F23C  00 00 00 00 00 00 00 00  00 00 00 00 00 00
0095F24A  00 00 00 00 00 00 00 00  00 00 00 00 00 00
0095F258  A9 00 00 00 00 00 00 00  C4 00 00 00 00 00
0095F266  00 00 D6 1D 27 00 00 00  00 00 07 0[illegible] 05 00
```

Beginning of UMGR32.exe

DirectoryTableBase offset
It stored the value for CR3 register.
So for UMGR32.exe, the CR3 i
0x075A700 (read from the left)

ProcessID (pid)
By converting this value to decimal, the pid of the process can be obtained
0x029c = 668 (decimal)

ActiveProcessLink offset
It stored the FLink and BLink of the EPROCESS. The first four offsets define FLink while the remainder is for Blink
FLink = 0xFF191EE0
BLink = 0xFF191CE0
(both are virtual addresses)

PEB (process environment bloc
offset
The value stored is virtual addr
of peb block (read from the left
0x7ffdf000

ImageFileName
UMGR32.EXE
offset from the beginning of the
EPROCESS block = 0x1fc

Fig 13: The information that is obtained from the value stored in some internal structures.

DirectoryTableBase represents CR3 register which is an important component in address translation to translate the virtual address of the structures into physical address. On the other hand, the ProcessID (PID) represents unique identifier of the process which is useful in verification of other block that is link with the process block. This is due to the fact that other object that is linked with the process block will inherit this value with them. Thus, PID value can be used to trace the hidden object by comparison test. Finally, most values that are stored in the EPROCESS block represent the virtual address for other block such as THREAD, PEB, HandleTable, and ActiveProcessLinks (e.g. the value stored in ThreadListHead point to virtual address of the first thread of the UMGR32.exe process block).

Although the test 3 is performed to retrieve the physical address for PEB of UMGR32.exe process, the step that has been discussed in the previous chapter can also be applied to obtain the physical address of any interested internal structure. Even though this algorithm can be used to retrieve the physical address of the internal structure, there is also a drawback that the author identified. This drawback is with regard to the existence of the Page Directory and Page Table Entries. In theory, this algorithm needs both entries to be resided in the physical memory. If one of the entries does not reside in the physical memory, the algorithm will be terminated. Hence, this algorithm can only work to retrieve the physical address of internal structure of active process alone. If the process is hidden or/and terminated, there is possibility that this algorithm will not work due to the factor that either one or both entries does not exist in the memory.

Thus, in order to minimize the drawback in the algorithm, the structure of virtual memory (e.g pagefiles) must be studied and then used that knowledge for combination with the algorithm to optimize the result.

6.0 Conclusion

As discussed in this paper, it is noted that most of the past existing tools were developed to solve or analyze the memory dump of a specific Windows version.

However, the author realized that although the offset of the internal structure in the process block for Windows 2000 and Windows XP are different from each other, the mechanism of retrieving the information remains the same for both operating systems. The internal structure known as DirectoryBaseTable is a critical component for address translation to obtain information from structures such as Process Environment Block (PEB), ETHREAD, Handle Table and others. In addition, the CreateTime and ExitTime in the process block are critical in forensics investigation as they represent the time length of the process being run in the computer.

Further, the investigator needs to know the architecture of the system before the address translation algorithms can be applied to trace the internal structure. This is due to the factor that different system architecture will apply different algorithm for translating the virtual address into physical address.

Acknowledgment

I would like to present my gratitude towards Technology PETRONAS University for the help and support that has been given during the research period.

References

[1] Symantec, N. f. *What is Cybercrime?* ,

[2] Hill, C. E. *"What is the Definition of Digital Forensics? "*.

[3] Carrier, B. E. H. Getting Physical with the Digital Investigation Process. *InternationalJournal of Digital Evidence*, 2, 2 Fall 2003).

[4] DFRWS *DFRWS 2005 Forensics Challenge*. 2005.

[5] DFRWS *Memparser Analysis Tool by Chris Betz*. 2005.

[6] DFRWS *Kntlist Analysis Tool by George M. Garner Jr.*, 2005.

[7] Jesee, K. Using every part of the buffalo in Windows memory analysis. *Digital Investigation*, 42007), 24-29.

[8] Mark E. Russinovich, D. A. S. *Microsoft® Windows® Internals, Fourth Edition: Microsoft Windows Server™ 2003, Windows XP, and Windows 2000*. Microsoft Press, 2004.

[9] Dhamdhere, D. M. *Operating Systems: A Concept based Approach*. McGrawHill, 2009.

[10] Russinovich, M. E., Solomon, D. A. and Ionescu, A. *Windows®Internals Covering Windows Server® 2008 and Windows Vista®*. Microsoft Press, City, 2009.

[11] Amari, K. *Techniques and Tools for Recovering and Analyzing Data from Volatile Memory*. SANS Institute, 2009.

[12] Carrier, B. G., J. A Hardware Based Memory Acquisition Procedure for Digital Investigations. *Journal of Digital Investigation*2004, March).

[13] Schuster, A. Searching for processes and threads in Microsoft Windows memory dump. *Digital Investigation*, 32006), 10-16.

[14] Dolan-Gavitt, B. The VAD tree: A process eye view of physical memory. *Digital Investigation*2007), s62-s64.

[15] Jr, G. M. G. *Forensic Acquisition Utilities*. 2009.

[16] Kornblum, E. L. a. J. D. A proposal for an integrated memory acquisition. *ACM SIGOPS Operating System, Computer Forensics*, 42, 3 2008), 14-20.

[17] Garfinkel, T., Pfaff, B., Chow, J., & Rosenblum, M. lifetime is a systems problem. In *Proceedings of the ACM SIGOPS European Workshop* (2004). ACM,

[18] Stevens, D., 2007, January 30.

[19] R.B. van Baar*, W. A., A.R. van Ballegooij Forensic memory analysis: Files mapped in memory. *Digital Investigation* 52008), 52-57.

[20] Corporation, A. *Importance of memory Search and Analysis (White Paper)*.

[21] Burdach, M. *An Introduction to Windows memory forensic*. 2005.

[22] Ruichao Zhang, L. W., Shuhui Zhang. Windows Memory Analysis Based on KPCR. In *Proceedings of the 2009 Fifth International Conference on Information Assurance and Security* (Xi'An China, 2009). IEEE

[23] S. M. Hejazi, C. T., M. Debbabi Extraction of forensically sensitive information from windows physical memory. *d i g i t a l i n v e s t i g a t i o n* 62009), S 1 2 1 – S 1 3 1.

[24] Schuster, A. *PTFinder*. 2006.

[25] Schuster, A.,