Network Connections Information Extraction of 64-Bit Windows 7 Memory Images

Lianhai Wang*, Lijuan Xu, and Shuhui Zhang

Shandong Provincial Key Laboratory of Computer Network,
Shandong Computer Science Center,
19 Keyuan Road, Jinan 250014, P.R. China
{wanglh,xulj,zhangshh}@Keylab.net

Abstract. Memory analysis technique is a key element of computer live forensics, and how to get status information of network connections is one of the difficulties of memory analysis and plays an important roles in identifying attack sources. It is more difficult to find the drivers and get network connections information from a 64-bit win7 memory image file than its from a 32-bit operating system memory image file. In a this paper, We will describe the approachs to find drivers and get network connection information from windows 7 memory images. This method is reliable and efficient. It is verified on Windows version 6.1.7600.

Keywords: computer forensics, computer live forensics, memory analysis, digital forensics.

1 Introduction

Computer technology has greatly promoted the progress of human society. Meanwhile, it also brought the issue of computer related crimes such as hacking, phishing, online pornography, etc. Now, computer forensics has emerged as a distinct discipline of knowledge in response to the increasing occurrence of computer involvement in criminal activities, both as a tool of crime and as an object of crime, and live forensics gains a weight in the area of computer forensics. Live forensics gathers data from running systems, that is to say, collects possible evidence in real time from memory and other storage media, while desktop omputers and servers are running. Physical memory of a computer can be a very useful yet challenging resource for the collection of digital evidence. It contains details of volatile data such as running processes, logged-in users, current network connections, users' sessions, drivers, open files, etc. In some cases, such as encrypted file systems arrive on the scene, the only chance to collect valuable forensic evidence is through physical memory of the computer. We propose a model of computer live forensics based on recent achievements of analysis techniques of physical memory image[1]. The idea is to gather "live" computer evidence through analyzing the raw image of target computer. See Fig. 1. Memory analysis technique is a key element of the model.

* Supported by Shandong Natural Science Foundation (Grant No. Y2008G35).

X. Lai et al. (Eds.): E-Forensics 2010, LNICST 56, pp. 90–98, 2011.

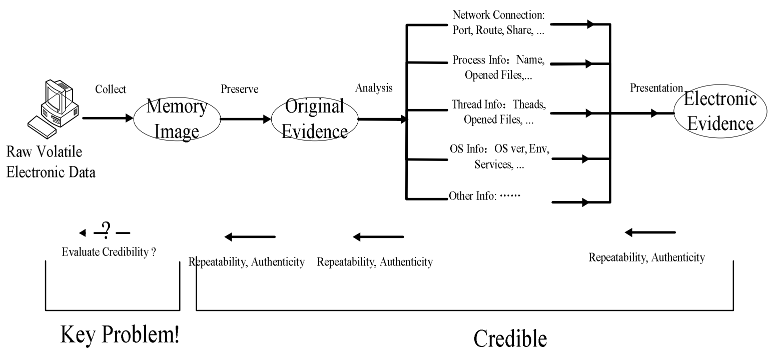

Fig. 1. Model of Computer Live Forensics Based on Physical Memory Analysis

How to get status information of network connections is one of the difficulties of memory analysis and plays an important roles in identifying attack sources. But it is more difficult to get network connections information from a 64-bit win7 memory image file than its from a 32-bit operating system memory image file. There are many difference bewetten the methods for 64-bit system and the method for 32-bit system. We will describe the approachs to get network connection information from 64-bit windows 7 memory images.

2 Related Work

In 2005, the Digital Forensic Research Workshop (DFRWS) organized a challenge of memory analysis (http://dfrws.org/2005/). And then Capture and analysis of the content of physical memory, known as memory forensics, became an area of intense research and experimentation. In 2006, A. Schuster analyzed the in-memory structures and developed search patterns which will then be used to scan the whole memory dump for traces of both linked and unlinked objects [2]. M. Burdach also developed WMFT (Windows Memory Forensics Toolkit) and gave a procedure to enumerate processes [3, 4]. Similar techniques in these works were also being used by A. Walters in developing Volatility tool to analyze memory dumps for an incident response perspective [5]. There are many others articles talked about memory analysis.

Nowadays, there are two methods to acquire network connection status information from physical memory of Windows XP operating system. One is searching for data structure "AddrObjTable" and "ObjTable" from driver "tcpip.sys" to acquire network connection status information. This method is implemented in Volatility[6], a tool to analyze memory which dumps from Windows XP SP2 or Windows XP SP3 for an incident response perpective developed by Walters and Petroni. The other one is proposed by Schuster[7]. Schuster descirbes the steps necessary to detect traces of network activity in a memory dump.His method is searching for pool allocations labeled "TcpA" and a size of 368 bytes (360 bytes for the payload and 8 for the _POOL_HEADER) on Windows XP SP2. These allocations will reside in the non-paged pool.

The first method is feasible on Windows XP. But it doesn't work on Windows Vista and Win 7 ,because there is no data structure "AddrObjTable" or "ObjTable" in driver "tcpip.sys". It is proven that there is no pool allocations labeled "TcpA" on Windows 7 as well.

It is analyzed that there are pool allocations labeled "TcpE" instead of "TcpA" indicating network activity in a memory dump of Windows 7. Therefore, we can acquire network connections from pool allocations labeled "TcpE" on Windows 7. This paper proposes a method of acquiring current network connection informations from physical memory image of Windows 7 according to memory pool. Network connection informations including IDs of processes which established connections, local address, local port, remote address, remote port, etc., can be get accurately from physical memory image file of Windows 7 with this method.

3 A Method of Network Connections Information Extraction from Windows 7 Physical Memory Images

3.1 The Structure of TcpEndpointPool

A data structure called TcpEndpointPool is found in driver "tcpip.sys" on Windows 7 operating system, and it is similar to its on Windows vista. This pool is a doubly-linked list of which each node is the head of a singly-linked list.

The internal organizational structure of TcpEndpointPool is shown by figure1. The circles represent heads of the singly-linked list. The letters in the circles represent the flag of the head. The rectangles represent the nodes of singly-linked list. The letters in the rectangles represent the type of the node.

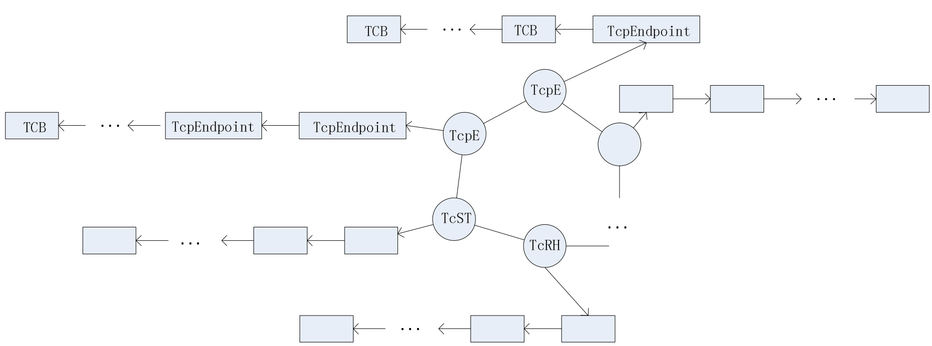

Fig. 2. TcpEndpointPool internal organization

The structure of singly-linked list head is shown by figure 2, in which there is a _LIST_ENTRY structure at the offset 0x40 by which the next head of a singly-linked list can be found .

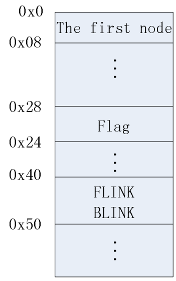

Fig. 3. The structure of singly-linked list head

The relationship of two adjacent heads is shown by figure 4.

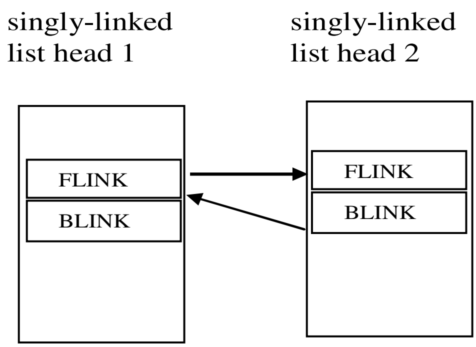

Fig. 4. The linked relationship of two heads

There is a flag at the offset 0x28 of the singly-linked list head by which the node structure of the singly-linked list can be judged. If the flag is "TcpE", the singly-linked list with this head is composed of TcpEndPoint structure and TCB structure which describe the network connection information.

3.2 The Structure of TCB

TCB Structure under Windows 7 is quite different form its under Windows Vista or XP. The definition and the offsets of fields related with network connections in the TCB is shown as follows.

```
typedef struct _TCB {
    CONST NL_PATH *Path;                 +0x30
    USHORT TcbState;                     +0x78
    USHORT EndpointPort                  +0x7a
    USHORT LocalPort;                    +0x7c
    USHORT RemotePort;                   +0x7e
    PEPROCESS OwningProcess ;            +0x238
    } TCB,*PTCB;
```

NL_PATH structure, NL_LOCAL_ADDRESS structure and NL_ADDRESS_ IDENTIFIER structure are defined as follows by which network connection local address and remote address can be acquried.

```
  typedef struct _NL_PATH {
     CONST NL_LOCAL_ADDRESS *SourceAddress;            +0x00
     CONST UCHAR *DestinationAddress;                  +0x10
 } NL_PATH, *PNL_PATH;
 typedef struct _NL_LOCAL_ADDRESS {
     ULONG Signature  // Ipla（0x49706c61）
     CONST NL_ADDRESS_IDENTIFIER *Identifier;          +0x10
 } NL_LOCAL_ADDRESS, *PNL_LOCAL_ADDRESS;
 typedef struct _NL_ADDRESS_IDENTIFIER {
     CONST UCHAR *Address;                             +0x00
 } NL_ADDRESS_IDENTIFIER, *PNL_ADDRESS_IDENTIFIER;
```

3.3 Algorithms

The algorithm to find all of TcpE pools is given as follows:

Step1. Get the physical address of KPCR structure and achieve the function of translation from virtual Address to physical address.

Because address stored in image file generally is virtual address, we can not directly get the exact location of its physical address in memory image file via its virutal address . First of all, we should achieve the function of translation from virtual Address to physical address ,which is a difficult problem in memory ananlsis. We can adopt a method, which is similar to the KPCR method[8], to achieve the function ,but It require change as show below:

I) Find KPCR structure according to characteristics as blow: find the two neighboring values is greater than 0xffff000000000000, and the difference between these two values is 0x180, Take away 0x1c from the phyical address of the first value , and we get the KPCR structure address.

II) The offset of CR3 Registe is not 0x410, but 0x1d0.

Step 2. Find dirvers of system ,and get the address of TCPIP.SYS driver

As a 64-bit operating system , it is more difficult to find the drivers of system from a 64-bit win7 memory image file than its from a 32-bit operating system memory image file. In Windows 7 system, KdVersionBlock,a elements of the structure KPCR, is always is zero, so we can’t get kernel variables thought it. We find a way to get the dirvers of system as blow:

Step2.1 Locate the address of KPRCB structure

the KPCR structure address add 0x180 ,we will get the address of _KPRCB structure.

```
 _KPCR{
     +0x108 KdVersionBlock    : Ptr64 Void
     +0x180 Prcb              : _KPRCB
 }
```

Step2.2 Locate the address of pointer pointed to the current thread

CurrentThread ,which is pointed the current thread of system, is a address pointer pointed a KTHREAD structure, and it is stored at the offset 0x08 relative to KPRCB structure address. We can get the phyical address which is pointed by the pointer according to the translation described as Step1

```
_KPRCB{
  +0x008 CurrentThread     : Ptr64 _KTHREAD
}
```

Step2.3 Locate the address of pointer of current process according to the current thread.

The virtual address of current process is stored at the offset 0x210 relative to KTHREAD structure. We will get the phyical address of current process from the virtual address according to the translation.

```
_KTHREAD{
+0x210 Process              : Ptr64 _KPROCESS
}
```

Step 2.4 Locate the address of ActiveProcessLinks

```
_EPROCESS{
  +0x000 Pcb                      : _KPROCESS
  +0x188 ActiveProcessLinks : _LIST_ENTRY
}
```

Step 2.5 Locate the address of the nt!PsActiveProcessHead variable

ActiveProcessLinks is the active process links, Throught it, we can get all of process. When we can the address of system process, we can the the address of the nt!PsActiveProcessHead variable from Blink of its ActiveProcessLinks .

```
_LIST_ENTRY{
  +0x000 Flink             : Ptr64 _LIST_ENTRY
  +0x008 Blink             : Ptr64 _LIST_ENTRY
}
```

Step 2.6 Locate the address of kernel variable psLoadedModuleList

The offset bewteen the virtual address of nt!psLoadedModuleList and the virtual address of nt!PsActiveProcessHead is 0x1e320, so the address of nt!PsActiveProcessHead add 0x1e320, we get the virtual address of nt!psLoadedModuleList. We get the physical address of nt!psLoadedModuleList according to the translation.

Step 2.7 Get the address of TCPIP.SYS driver through the kernel variable psLoadedModuleList.

Step3 Find the virtual address of tcipip!TcpEndpointPool.

We can get the virtual address of tcpip!TcpEndpointPool from the virutal address added 0x18a538.

Step4 Find the virtual address of the first singly-linked list head.

Firstly, transfer the virtual address of TcpEndpointPool to physical address and locate the address in the memory image file, read 8 bytes at this position and transfer the 8 bytes to physical address, locate the address in the memory image file. Secondly , get the the virtual address of the pointer which is the 8 bytes at the offset 0x20 . this pointer points three virtual address pointer pointed the structures in which singly-linked list head is the 8 bytes at the offset 0x40.

The search process on Windbg can be shown in Fig.5

```
fffff880`0198d538 tcpip!TcpEndpointPool = <no type information>
lkd> db  tcpip!TcpEndpointPool
fffff880`0198d538  40 9a 47 04 80 fa ff ff-00 00 00 00 00 00 00 00  @.G.............
fffff880`0198d548  a0 9a 86 01 80 f8 ff ff-00 00 00 00 00 00 00 00  ................
fffff880`0198d558  00 00 00 00 00 00 00 00-a0 ba 47 04 80 fa ff ff  ..........G.....
fffff880`0198d568  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
fffff880`0198d578  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
fffff880`0198d588  50 a3 86 01 80 f8 ff ff-00 00 00 00 00 00 00 00  P...............
fffff880`0198d598  00 00 00 00 00 00 00 00-00 ba 47 04 80 fa ff ff  ..........G.....
fffff880`0198d5a8  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
lkd> db fffffa8004479a40
fffffa80`04479a40  03 00 00 00 00 00 00 00-54 63 70 45 54 63 70 45  ........TcpETcpE
fffffa80`04479a50  10 03 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
fffffa80`04479a60  40 88 47 04 80 fa ff ff-00 00 00 00 00 00 00 00  @.G.............
fffffa80`04479a70  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
fffffa80`04479a80  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
fffffa80`04479a90  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
fffffa80`04479aa0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
fffffa80`04479ab0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
lkd> db fffffa8004478840
fffffa80`04478840  40 99 47 04 80 fa ff ff-40 98 47 04 80 fa ff ff  @.G.....@.G.....
fffffa80`04478850  00 bb 47 04 80 fa ff ff-00 00 00 00 00 00 00 00  ..G.............
fffffa80`04478860  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
fffffa80`04478870  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
fffffa80`04478880  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
fffffa80`04478890  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
fffffa80`044788a0  08 00 10 02 54 63 44 4d-00 00 00 00 00 00 00 00  ....TcDM........
fffffa80`044788b0  01 00 0f 06 54 63 44 4d-00 00 00 00 00 00 00 00  ....TcDM........
lkd> db fffffa8004479940
fffffa80`04479940  02 00 fb 00 00 00 00 00-71 67 81 06 80 fa ff ff  ........qg......
fffffa80`04479950  04 00 00 01 b1 02 00 00-b8 01 00 00 85 02 00 00  ................
fffffa80`04479960  8a 01 00 00 00 00 00 00-54 63 70 45 10 03 00 00  ........TcpE....
fffffa80`04479970  50 66 e9 03 00 f8 ff ff-ec b1 e9 03 00 f8 ff ff  Pf..............
fffffa80`04479980  80 98 47 04 80 fa ff ff-40 9b 47 04 80 fa ff ff  ..G.....@.G.....
fffffa80`04479990  b1 02 00 00 b8 01 00 00-00 00 00 00 00 00 00 00  ................
fffffa80`044799a0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
fffffa80`044799b0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
```

Fig. 5. The process to find the virtual address of the first singly-linked list head on Windbg

Step5 Judge whether the head's type is TcpEndpoint or not by reading the flag which is set at the offset 0x20 relative to the head's address. If the flag is "TcpE", the head's type is TcpEndpoint , go to the step 6, otherwise go to the step 7.

Step6 Analyze the TcpEndpoint structure or TCB structure in the singly-linked list. Analyzing algorithm is shown by figure 6.

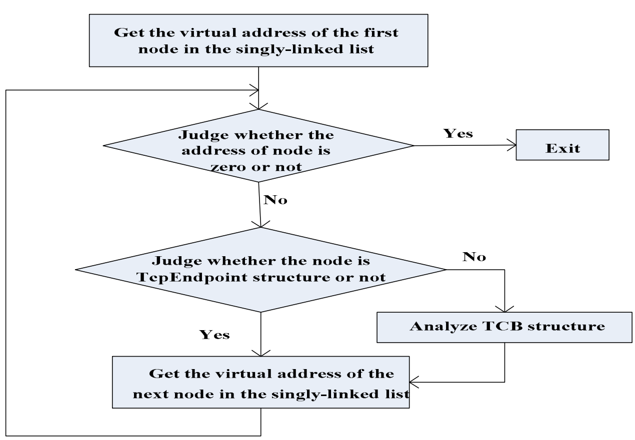

Fig. 6. The flow of analyzing TCB structure or TcpEndpoint structure summary description

Step7 Find the virtual address of the next head.

The virtual address of the next head can be found according to the _LIST_ENTRY structure which is set at the offset 0x30 relative to the address of singly-linked list head. Judging whether the next head's virtual address equals to the first head's address or not. If the next head's virtual address is equal to the first head's address, exit the procedure, otherwise go to the next step.

Step8 Judge whether the head is exactly the first head. If the head is exactly the first head, exit, otherwise go to step 5.

The flow of analyzing TCB structure or TcpEndpoint structure is shown as follows.

Step1 Get the virtual address of the first node in the singly-linked list.

Transfer the virtual address of singly-list head to physical address and locate the address in memory image file. Read 8 bytes from this position which is the virtual address of the first node.

Step2 Judge whether the address of node is zero or not. If the address is zero, exit the procedure, otherwise go to the next step.

Step3 Judge whether the node is Tcb structure or not.

if LocalPort#0 and RemotePort#0 then it is a TCB Structure , furthermore, if TcbState#0 it is valid TCB Structure ,or it is a tcb structure which it indicate the network connection is close.

if LocalPort=0 and RemotePort=0 and EndpointPort#0 then it is a TCP_ENDPOINT structure

Step4 Analyze TCB structure.

Step4.1 Get PID (process id) which is the ID of the process which established this connection. The pointer which points to the process's EPROCESS structure which established this connection is set at the offset +0x238 relative to TCB structure. Firstly, read 8 bytes which represents the virtual address of EPROCESS structure at buffer's offset 0x164 and transfer it to physical address. Secondly, locate the address in the memory image file and read 8 bytes which represents PID at the offset 0x180 relative to EPROCESS structure's physical address.

Step4.3 Get the local port of this connection. The number is set at offset 0x7c of TCB structure. Read 2 bytes at offset 0x7C of the buffer and transfer it to a decimal which is the local port of this connection.

Step4.4 Get the remote port of this connection. The number is set at the offset 0x7e of TCB structure. Read 2 bytes at offset 0x7e of the buffer and transfer it to a decimal which is the remote port of this connection.

Step4.5 Get local address and remote address of this connection. The pointer which points to NL_PATH structure is set at the offset 0x30 of TCB structure. The pointer which points to the remote address is set at the offset 0x10 of NL_PATH structure. The special algorithm is as followes: read 8 bytes which represents the virtual address of NL_PATH structure at the offset 0x30 of TCB structure, transfer the virtual address of NL_PATH structure to physical address, locate the address+0x10 in the memory image file and read 8 bytes which represents remote address at this position. The pointer which points to NL_LOCAL_ADDRESS structure is set at the offset 0x0 of the NL_PATH structure, The pointer which points to NL_ADDRESS_IDENTIFIER structure is set at the offset 0x10 of

NL_LOCAL_ADDRESS structure, local address is set at the offset 0x0 of the NL_ADDRESS_IDENTIFIER structure. Therefore, local address can be acquired from the above three structures.

Step5 Get 8 bytes which represents the next node's virtual at the offset 0 of the buffer and go to step2.

4 Conclusion

In this paper, a method which can acquire network connection information from 64-bit Windows 7 memory image file based on memory pool allocation strategy is proposed. This method is proved to be right for memory image file of Windows version 6.1.7600. This method is reliable and efficient, because the data structure TcpEndpointPool exists in driver tcpip.sys for different Win7 operation system versions and TcpEndpointPool structure will not change when Win 7 operation system version changed.

References

1. Wang, L., Zhang, R., Zhang, S.: A Model of Computer Live Forensics Based on Physical Memory Analysis. In: ICISE 2009, Nanjing China (December 2009)
2. Schuster, A.: Searching for Processes and Threads in Microsoft Windows Memory Dumps. In: Proceedings of the 2006 Digital Forensic Research Workshop, DFRWS (2006)
3. Burdach, M.: An Introduction to Windows Memory Forensic[OL] (July 2005), `http://forensic.seccure.net/pdf/introduction_to_windows_memory_forensic.pdf`
4. Burdachz, M.: Digital Forensics of the Physical Memory [OL] (March 2005), `http://forensic.seccure.net/pdf/mburdach_digital_forensics_of_physical_memory.pdf`
5. Walters, A., Petronni Jr., N.L.: Volatools: Integrating volatile Memory Forensics into the Digital Investigation Process. In: Black Hat DC (2007)
6. Volatile Systems: The Volatility Framework: Volatile memory artifact extraction utility framework (accessed, June 2009), `https://www.volatilesystems.com/default/volatility/`
7. Andreas, S.: Pool allocations as an information source in windows memory forensics. In: Oliver, G., Dirk, S., Sandra, F., Hardo, H., Detlef, G., Jens, N. (eds.) IT-incident management & IT-forensics-IMF 2006, October 18. Lecture notes in informatics, vol. P-97, pp. 104–115 (2006b)
8. Zhang, R., Wang, L., Zhang, S.: Windows Memory Analysis Based on KPCR. In: Fifth International Conference on Information Assurance and Security, IAS 2009, vol. 2, pp. 677–680 (2009)