Live Memory Acquisition through FireWire

Lei Zhang, Lianhai Wang, Ruichao Zhang, Shuhui Zhang, and Yang Zhou

Shandong Provincial Key Laboratory of Computer Network,
Shandong Computer Science Center,
19 Keyuan Road, 250014 Jinan, Shandong, China
{zhanglei,wanglh,zhangrch,zhangshh,zhouy}@keylab.net

Abstract. Although FireWire-based memory acquisition method has been introduced for several years, the methodologies are not discussed in detail and still lack of practical tools. Besides, the existing method is not working stably when dealing with different versions of Windows. In this paper, we try to compare different memory acquisition methods and discuss their virtues and disadvantages. Then, the methodologies of FireWire-based memory acquisition are discussed. Finally, we give a practical implementation of FireWire-based acquisition tool that can work well with different versions of Windows without causing BSoD problems.

Keywords: live forensics; memory acquisition; FireWire; memory analysis; Windows registry.

1 Introduction

Live memory forensics, typically consists of live memory acquisition and memory analysis, is playing a more and more important role in modern computer forensics because of "in memory only" malwares, widely using of file and disk encrypting tools [1], and a lot of useful information that resides only in system memory and can't be acquired through traditional forensics methods [2].

To acquire volatile system memory, there are mainly two different ways, hardware-based and software-based [3]. Software-based methods are widely used because of their simplicity and freeness - many memory acquisition tools are available on internet and can be downloaded freely. This results in a boom of live memory forensics technologies.

Despite the virtues, software-based methods can not deal with locked systems when the unlock password is unknown since they need to run software application program(s) on the subject machine. At the same time, running of such software acquisition tools needs to use relative large memory (compared to hardware-based methods) of the subject system, this may overwrite useful data and destroy the integrity of system memory data and keep it from being evidence. Moreover, software based memory acquisition tools can be easily cheated by anti-forensic malwares since running of these tools is heavily based on services provided by the subject system OS which may have been manipulated by these malwares.

Hardware based memory acquisition tools could be used to resolve these problems or just to improve the performance, these tools typically do memory acquisition work

X. Lai et al. (Eds.): E-Forensics 2010, LNICST 56, pp. 159–167, 2011.

in DMA (Direct Memory Access) mode, by this way, the subject system OS is bypassed when they are working. At the same time, these methods do not need to run any software application in the subject system.

So far, there are two different hardware based methods to acquire system memory, one is using a PCI expansion card, the other is through a FireWire port. The PCI-card method needs a pre-installation of the acquisition card into the subject system before incidents happen, this narrows its usability. FireWire, also called IEEE 1394, is shipped with many modern notebooks or even desktop computers. Even if there are no FireWire ports directly equipped on the machine, they could be expanded through PCMCIA or PCI Express expansion cards. As the subject system OS is bypassed when these acquisition tools are accessing system memory in DMA mode, password is unneeded to dump system memory out of the locked machine. But how could FireWire-based tools get the right to access system memory, and what steps should be taken to dump the whole system memory? In this paper, we will discuss these problems and give an implementation of the FireWire-base memory acquisition method, and this tool can work stably with Windows operating systems.

The rest of this paper is organized as follows. Section 2 discusses base concepts of live memory acquisition and compare different acquisition methods. Section 3 discusses methodologies of FireWire-based memory acquisition and give a practical implementation of this method. Section 4 discusses what we can do in the future. Section 5 is the conclusion of this paper.

2 Live Memory Acquisition, Methods and Available Tools

Traditional computer forensics, also called “static forensics”, is mainly based on static disk images acquired from a “dead” machine. There are many problems such as shut down process, unreadable encrypted data, and incomplete evidence [4] by using this traditional method. Live memory forensics could be used to try to resolve these problems.

Live memory acquisition, being the first step of memory forensics, is performed on a running subject machine. There are mainly two different ways to acquire system memory, software-based and hardware-based. A set of tools are associated to each of them. In this section, we will discuss virtues and limitations of each of them.

2.1 Software-Based Acquisition

System memory is managed as a special device in many modern operating systems. Table 1 shows the device name and user mode availability in different operating systems.

Table 1. Physical memory device name and availability in different operating systems

Operating system	Physical memory device	Availability in user mode
UNIX	/dev/mem	Available
Linux	/dev/mem	Available
MAC OS X	/dev/mem	Not available
Windows	\device\PhysicalMemory	Not available since Windows 2003 SP1

There are a set of software tools such as "*dd*", "*mdd*", "*Nigilant32*", "*Win32dd*", "*nc*", "*F-Response*", and "*HBGary FastDump*" that could be used to dump physical memory out from subject systems. As an example, the physical memory could be dumped through a simple command line by "*dd*":

```
dd if=/dev/mem of=mymem.img conv=noerror,sync
```

The physical memory could also be dumped to a remote system by "*nc*", the command line is listed below:

```
nc -v -n -I \\.\PhysicalMemory <ip> <port>
```

These software acquisition tools are very easy to use and can be downloaded from internet freely, but they also have many limitations such as need a full control right of the subject system and have relatively heavy footprint since they must be loaded into the subject system memory and running there. For Windows operating systems after Windows 2003 SP1, the *\\.\PhysicalMemory* device is not available in user mode, thus memory acquisition tools that use this device and run in user mode can't work anymore. Moreover, these tools are based on services provided by the subject OS, so they could be easily cheated by anti-forensic malwares.

2.2 Hardware-Based Acquisition

Hardware-based memory acquisition tools are not that popular as software ones because they need additional hardware devices. The hardware device, in forms of a PCI expansion card, a dedicated Linux-based machine or a special-designed hardware is either very expensive or just not available on general markets. These tools, either pre-equipped or post-installed, could be attached to subject systems and dump the system memory in DMA mode. These tools need not to run any software agent in the subject system and could circumvent the subject system OS when they are working. Thus they could hardly be cheated by anti-forensic malwares (But also could be defeated by changing settings of registers in the North Bridge [5]) and have relatively light footprint in the subject system memory. There are typically two different kinds of hardware-based memory acquisition methods, one is through PCI bus, the other is through FireWire ports.

As to PCI bus method, a tool named "*Tribble*" [6] is introduced in February 2004 by *Brain Carrier*, et.al. This method uses a pre-installed PCI expansion card to acquire system memory when incidents happen. With a switch being turned on to start the dumping process, "*Tribble*" does not introduce any software to the subject system thus it has a good performance on protecting data integrity. But, the need of pre-installing of the acquisition card heavily limits its usage.

FireWire began to attract forensic experts' attention as a memory acquisition tool after the initial introduction as a way to hack into locked systems by the use of a modified "ipod" [7] in 2005. This method can only acquire memory of Linux-based systems until 2006, when Adam Boileau first gave a method to cheat the target Windows-based OS to give the acquisition tool Direct Memory Access right [8]. This method does not need any pre-installation. FireWire ports are equipped with many modern computers, even if there is not such a port that already integrated on the system motherboard, it could be expanded through a PCMCIA or PCI Express slot.

Although this method has emerged and has been used by forensic experts for some years, there are still problems such as weak stability in dealing with Windows-based systems and might run into a BSoD (Blue Screen of Death) state when try to access the UMA (Upper Memory Area) [9] or other spaces that were not mapped into system memory. We will discuss methods of how to resolve these problems in section 3.

3 Methodologies and an Implementation of FireWire-Based Memory Acquisition

FireWire-based devices communicate to host computers through FireWire bus by using a protocol stack, the structure of this stack is shown in Figure 1. The IEEE 1394 protocol mainly specifies the physical layer electrical and mechanical characteristics, and it also defines link layer protocols of FireWire bus. The OHCI (Open Host Controller Interface) standard specifies the implementation of IEEE 1394 protocol in the "host computer" side. The transport protocols, such as SBP-2 (Serial Bus Protocol 2), define the protocol of transferring commands and data over FireWire bus. The device-type specific command sets, such as RBC (Reduced Block Commands and SPC-2 (SCSI Primary Commands - 2), define the commands that should be implemented by the device.

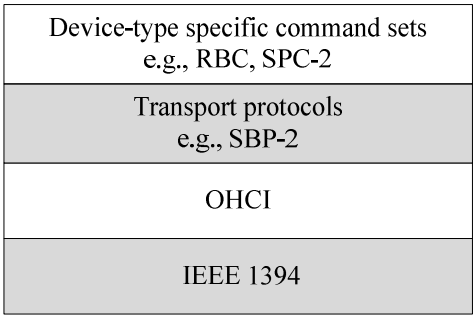

Fig. 1. Protocol stack of FireWire-based devices

To achieve best performance, the IEEE 1394 protocol gives the "target" device the ability to direct access system memory, by this way the host CPU could be freed from charging large amount of data transfers to or from system memory. According to IEEE 1394 protocol, read or write data packages are transferred from source nodes to destination nodes with a 64-bit destination address contained in these packages. The destination address consists of two parts, 16-bit *destination_ID* which consists of 10-bit bus address and 6-bit node address, and 48-bit *destination_offset*. The structure of a block read request package is shown in Figure 2.

The 16-bit *destination_ID* field contains the destination bus and node address, the 48-bit *destination_offset* is the destination address inside the target node. The OHCI standard gives an explaining of this 48-bit destination offset address. When the 48-bit address is below the address stored in the *Physical Upper Bound* register or less than

the default value 0x000100000000 if the *Physical Upper Bound* register is not implemented, the 48-bit target address will be explained by the host OHCI controller as a physical memory address, and then the OHCI controller will perform a direct memory transfer using the *Physical Response Unit* inside it. By this way the "target" device could address the host computer's system memory and perform both physical memory read and write transfers. By our testing and reading on datasheets of different OHCI controllers, the *Physical Upper Bound* register is either unimplemented or has a default value of all 0s, this will cause the OHCI controller to take a default value of 0x000100000000 as physical upper bound. Till now the acquisition tool already can deal with Linux and MAC OS X based systems, but not to Windows-based ones, why? According to OHCI standard, besides the *Physical Upper Bound* register, there are also another two registers that should be set correctly to make the read or write transfers be of sense. These two registers are *PhysicalRequestFiltersHi* and *PhysicalRequestFiltersLo*. Each bit in these two registers is associated with a device node indicated by the 6-bit node address in the *source-ID* field. When the associated bit is cleared to "0", the OHCI controller will forward the request to the *Asynchronous Receive Request DMA context* instead of *Physical Response Unit*, and this request will be processed by the associated device driver and the *destination_offset* will be explained as virtual memory address, thus the "target" device can't get the actual physical memory contents.

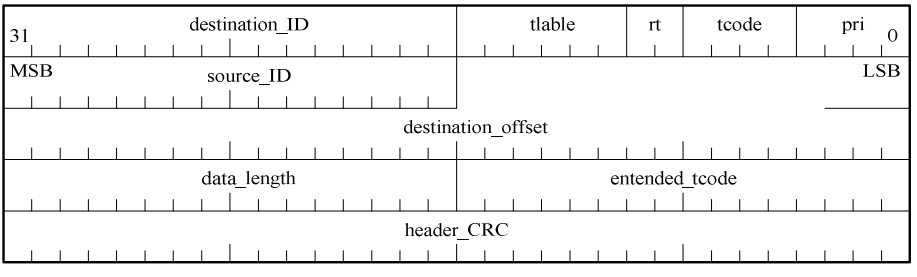

Fig. 2. Block read request package format

Fortunately, by the research of Adam Boileau, the physical DMA right could be gained if the "target" device pretends itself to be an ipod or a hard disk. By using the configure ROM of an ipod or hard disk, the "target" device could cheat the host computer to gain the DMA right. But, through our research, this method is not very stable towards different versions of Windows operating systems because of different implementations of file system drivers such as *disk.sys* and *partmgr.sys*. Since the file system is not implemented in the "target" device, it can't respond to commands sent from host computer, and to some versions of Windows, this will cause repeated sending of these commands and finally result in a bus reset with associated bit in the *PhysicalRequestFilterxx* registers being cleared to "0", this will prevent the acquisition tool from working. To resolve this problem, the mandatory commands associated with the device type given in the configure ROM should be implemented in the "target" device. The mandatory commands needed by a "Simplified direct-access" type device using a command set of RBC is listed in Table 2.

Table 2. Commands must be implemented in "Simplified direct-access" type devices

Command name	Opcode	Referenced command set
INQUIRY	12h	SPC-2
MODE SELECT	15h	SPC-2
MODE SENSE	1Ah	SPC-2
READ	28h	RBC
READ CAPACITY	25h	RBC
START STOP UNIT	1Bh	RBC
TEST UNIT READY	00h	SPC-2
VERIFY	2Fh	RBC
WRITE	2Ah	RBC
WRITE BUFFER	3Bh	SPC-2

Till now the acquisition tool could be attached to the host system and working stably. But, there is still another problem to acquire the whole subject system memory - since the length of the system memory is unknown, the acquisition tool does not know when to stop, and this may result into a BSoD state finally when the acquisition tool try to reading addresses not mapped into system memory. So the memory length information should be acquired before the address runs out of system memory range. To a subject system that in a locked state, the only information available is system memory, so the memory length information should be work out from the data stored in system memory.

As to a Windows operating system, the system registry is made up of a number of binary files called *hives*, among these *hives* there is a special one called *hardware* that stores information of hardware detected when the system was booting [10]. These information is only stored in system memory and thus could be acquired by the FireWire-based acquisition tool. There is a registry value named *.Translated* in the location of *HKEY_LOCAL_MACHIME/HARDWARE/RESOURCEMAP/System Resources/Physical Memory* in the *hardware hive* that stores base addresses and lengths of all memory segments. These memory segments could be accessed with no problem because they are mapped into truly physical memory. Figure 3 shows the *.Translated* registry value's contents, the *Physical Address* column shows the base addresses of different memory segments, and the *length* column shows the length of each memory segment. As an example, the "*0x001000*" in the *Physical Address* column is the base address of the first memory segment. The "*0x9e000*" in the *length* column is the first segment length. So, the address space of this memory segment is from *0x00001000* to *0x0009f000*. The first and last 4K bytes of the first 640K bytes system memory below UMA are not included in the first memory segment, but they could also be acquired properly. So we can use the first memory segment with its range from *0x00000000* to *0x000a0000*. We will use this fixed segment when we start memory acquisition work because the memory segments information is unknown in this stage. The second memory segment begins from the address *0x00100000*, between the first two segments is the UMA space. This space should be circumvented otherwise it may cause BSoD problem. In traditional computers, the memory space *0x00fff000-0x01000000* is used by some ISA cards and does not map into physical

memory, this generates a memory hole. To be compatible with traditional computers, this memory hole is maintained by modern operating systems though there are no ISA cards in the computer and this space is actually mapped into physical memory. So, this "hole" can be neglected because it does not actually exist. The next segment begins from *0x01000000* contains all the rest of the physical memory. So, we just have to bypass the UMA space before we find the memory segments information.

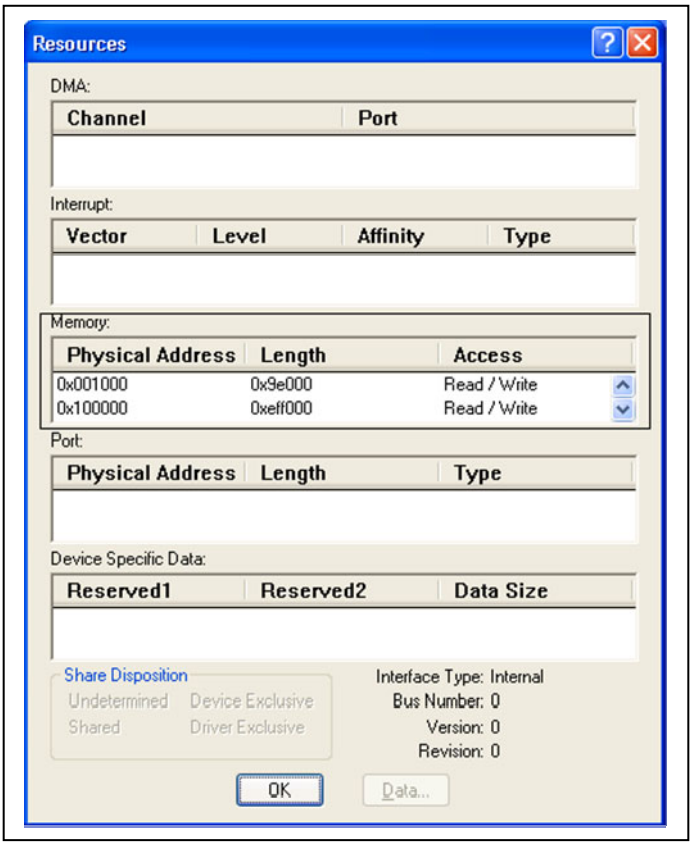

Fig. 3. Memory segments information contained in the *.Translated* registry value

The *.Translated* registry value data that stores in physical memory in a binary format is shown in Figure 4. So we can either search the registry value data using the character string "*.Translated*" or we can use the method provided by [10] to get this registry value data out from system memory.

Then, we could use the acquired information to generate base address and length of each memory segment. By this way, we never go into address spaces that are not mapped into physical memory thus the acquisition tool could work well without causing the target system to crash.

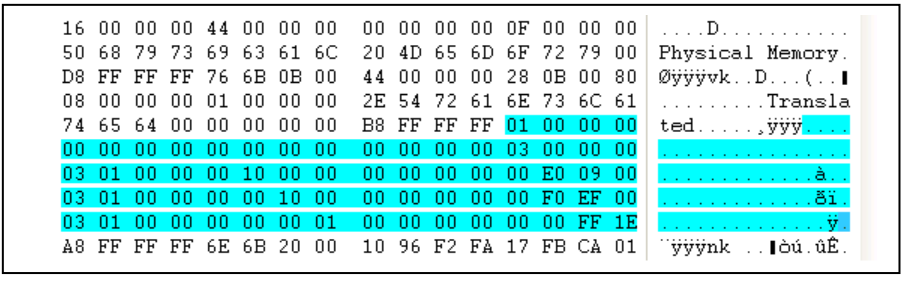

Fig. 4. Binary memory segments information stored in system memory

4 Future Work

Although OHCI protocol supports physical DMA in memory range over 4GB by properly setting the *Physical Upper Bound* register, most OHCI controllers do not support memory address longer than 32 bits because the *Physical Upper Bound* register is not implemented in them. Furthermore, even if this register is implemented in the OHCI controller, it can only be set by the OHCI controller driver from the host computer side and can't be accessed by the acquisition tool. So the amount of memory that FireWire-based acquisition tools can acquire is no more than 4GB. As for modern computers, the system memory becomes more and more large. Lots of computers have more than 4GB memory now, and modern operating systems are already capable of supporting systems with more than 4GB memory. So, how to get the memory over 4GB, and how to acquire the memory more rapidly? FireWire is not dependable because of its limitations. We have to look for substitute ways to resolve these problems. PCI Express bus, a serial version of the most popular used parallel PCI bus, has many new characteristics such as supporting hot-plug and supporting up to 64-bit memory address. The PCI Express bus is accessible from outside of a notebook through an Express card slot. Inserting a PCI Express add-in card to a "*live*" desktop or server may also be operable. So, we think the PCI Express-based memory acquisition tools may be the next step of hardware-based memory acquisition and will become available in the near future. Furthermore, because the memory contents keep changing while the acquisition tool is working, the consistency of the acquired data is not guaranteed. If the target system could be halted before acquisition work begins, the consistency of memory data will be protected. So methods of how to halt the target machine deserve further research.

5 Conclusion

In this paper, we discussed methodologies of FireWire-based memory acquisition and gave a method of how to get memory segment information from Windows registry to avoid access spaces that were not mapped into physical memory. We have worked out a proof-of-concept tool based on these methods, and now it can deal with Linux, MAC OS X, and almost all versions of Windows newer than Windows XP SP0. But

because of the limitations of FireWire, memory above 4GB can't be acquired, and the acquisition speed is relatively low. So substitute ways such as PCI Express bus should be considered in the future work.

Acknowledgement. We would like to express thanks to the following people who assisted in the proofing, testing and live demonstrations of the methods described above. Shandong Computer Science Center: Qiuxiang Guo, Shumian Yang and Lijuan Xu.

References

1. Casey, E.: The impact of full disk encryption on digital forensics. ACM SIGOPS Operating Systems Review 42(3), 93–98 (2008)
2. Brown, C.L.: Computer Evidence: Collection & Preservation. Charles River Media, Hingham (2005)
3. Ruff, N.: Windows memory forensics. Journal in Computer Virology 4(2), 83–100 (2008)
4. Hay, B., Bishop, M., Nance, K.: Live Analysis: Progress and Challenges. IEEE Security and Privacy 7, 30–37 (2009)
5. Rutkowska, J.: Beyond The CPU: Defeating Hardware Based RAM Acquisition Tools (Part I: AMD case), http://invisiblethings.org/papers/cheating-hardware-memoryacquisition-updated.ppt
6. Carrier, B., Grand, J.: A Hardware-based Memory Acquisition Procedure for Digital Investigations. Digital Investigation 1(1), 50–60 (2004)
7. Dornseif, M.: FireWire - all your memory are belong to us, http://md.hudora.de/presentations/
8. Boileau, A.: Hit by a Bus: Physical Access Attacks with FireWire. Security-Assessment.com, http://www.security-assessment.com/files/presentations/ab_firewire_rux2k6-final.pdf
9. Upper Memory Area Memory dumping over FireWire—UMA issues, http://ntsecurity.nu/onmymind/2006/2006-09-02.html
10. Dolan-Gavitt, B.: Forensic analysis of the Windows registry in memory. Digital Investigation 5(supplement 1), 26–32 (2008)