Reliable Acquisition of RAM Dumps from Intel-Based Apple Mac Computers over FireWire

Pavel Gladyshev[1] and Afrah Almansoori[2]

[1] Center for Cybercrime Investigation, University College Dublin
Belfield, Dublin 4, Ireland
Pavel.Gladyshev@ucd.ie
[2] Electronic Forensics Department, Dubai Police Head Quarters,
Dubai, United Arab Emirates
Almansoori@CyberCrimeTech.com

Abstract. RAM content acquisition is an important step in live forensic analysis of computer systems. FireWire offers an attractive way to acquire RAM content of Apple Mac computers equipped with a FireWire connection. However, the existing techniques for doing so require substantial knowledge of the target computer configuration and cannot be used reliably on a previously unknown computer in a crime scene. This paper proposes a novel method for acquiring RAM content of Apple Mac computers over FireWire, which automatically discovers necessary information about the target computer and can be used in the crime scene setting. As an application of the developed method, the techniques for recovery of AOL Instant Messenger (AIM) conversation fragments from RAM dumps are also discussed in this paper.

Keywords: RAM Analysis, Mac OS X, FireWire, AOL Instant Messenger (AIM).

1 Introduction

Forensic acquisition of data from a running computer system is commonly referred to as live forensics or computer forensics field triage. It is performed in situations when the extraction of evidential data through post mortem forensic analysis is either impossible or infeasible. One such situation is when the evidential data resides only in the RAM of the target computer, and would be lost if the computer was switched off.

Within this paper, the contributions to the field of live forensics are two-fold. Section 2 presents a FireWire based technique for reliably acquiring the contents of RAM whilst avoiding memory mapped input-output registers. This technique is applicable to Intel based Apple Mac computers running Mac OS X, and requires no prior knowledge of RAM size or the target system's memory map. Section 3 presents an application of this RAM imaging technique, to the task of AOL Instant Messenger (AIM) conversation fragment recovery from RAM; the process, methodology, and Perl scripts are explained in detail.

I. Baggili (Ed.): ICDF2C 2010, LNICST 53, pp. 55–64, 2011.

2 Reliable Acquisition of Mac OS X RAM Content via FireWire

2.1 State of the Arts

IEEE 1394 is a serial bus interface standard for high-speed data communication. Apple Computer Inc. uses brand name of FireWire for IEEE 1394 technology. One of the features of FireWire is that devices connected to the FireWire bus have access to each other's RAM. Several authors including M. Dornsief, et al [10], and A. Boileau [1,2] demonstrated the possibility of acquiring RAM content of the target computer over FireWire. To do so, the target computer is connected to the investigator's computer via FireWire. A special application is then executed on the investigator's computer to extract the data from the target computer's RAM. This paper refers to the acquisition of RAM content of the target computer as "RAM imaging".

The main problem with the existing approaches to FireWire RAM imaging is caused by the presence of memory mapped input-output device registers (or simply memory mapped I/O registers) in the memory address space of the target computer. Accidental reading of these registers over FireWire may trigger undesired activity in the corresponding devices and may cause instability and system crash in the target computer. The existing applications for FireWire RAM imaging, such as 1394memimage [2], need to be explicitly told which memory areas to image. To do so, the investigator needs to know the size of RAM in the target computer and the location of the memory mapped I/O register areas *apriori*. Although it is possible in incident response situations, where the investigator has the complete knowledge of the target system, it is not practical in live forensics situations, when a previously unknown live computer is found at the crime scene.

To avoid instability of the target system during FireWire RAM imaging at the crime scene, it is essential for the RAM imaging application to be able to automatically discover the location of the memory mapped I/O registers in the target computer. This research was able to develop a suitable process for Intel-based Apple Mac computers running Mac OS X operating system; this process is explained below.

2.2 Reliable FireWire RAM Imaging for Intel-Based Apple Mac Computers

The analysis of Mac OS X source code has shown that the information about the location of memory mapped I/O registers is stored in a special data structure, `MemoryMap`, of Mac OS X kernel. To access this data structure over FireWire, the RAM imaging application needs to identify

1. the version of Mac OS X on the target computer, and
2. the address of the `MemoryMap` data structure.

The following describes in detail the process and pitfalls in answering these questions.

Accessing Kernel Data Structures over FireWire. Our experiments have shown an investigator's computer connected to an Intel-based Apple Mac via FireWire can access the first 4GiB of the Apple Mac's physical address space[1]. The source code of Mac OS X kernel is freely available from Apple Open Source connection [5] and

[1] 4GiB limitation is caused by the 32-bit addressing of the host memory used in FireWire.

together with the debugging symbol information available in Apple's Kernel debugging kits, it can be used to interpret the content of the accessible Apple Mac RAM.

The method for interpreting the content of physical memory dumps of Apple Mac computers was first described by M. Suiche in [6]. In particular, it was shown that the virtual address of the *statically* allocated data structures in Mac OS X kernel is equal to the corresponding physical memory address. In relation to FireWire RAM imaging this means that if, for example, a statically allocated kernel data structure has virtual address 0x005db044, it can be accessed over FireWire using physical address 0x005db044. This rule does not apply to all kernel data, but it does apply to all data structures mentioned in this paper.

The address of the statically allocated data structure in a particular version of Mac OS X can be determined by examining its kernel executable file `mach_kernel` using, for example, `nm` command as shown in Fig. 1.

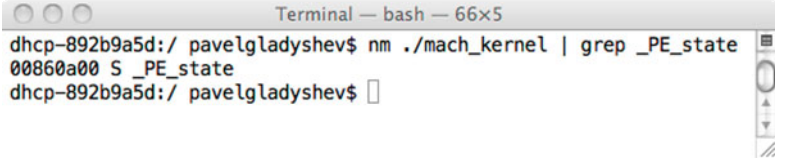

Fig. 1. Determining the address of a kernel data structure

The kernel executable files for different versions of Mac OS X can be downloaded from the Apple Developer Connection [7] as part of the kernel debugging kits.

Determining the Version of Mac OS X. Mac OS X kernel has a string constant called `version`, which is a null-terminated ASCII string describing the version of Mac OS X. E.g. "Darwin Kernel Version 10.0.0: Fri Jul 31 22:47:34 PDT 2009; …". The address of the `version` string is different in different versions of Mac OS X.

The version of Mac OS X running on the target computer can be determined by reading the contents of every possible location of the `version` string over FireWire and comparing the acquired data with the expected content of the `version` string. Once the `version` string for a particular version of Mac OS X is found at its expected location, it can be concluded that that version of Mac OS X is running on the target computer.

Determining Safe Memory Regions for Imaging. Intel-based Apple Mac computers implement Extendible Firmware Interface (EFI) standard [8]. According to the EFI specification, computer firmware communicates the layout of memory mapped I/O registers and other areas of memory to the operating system at boot time. Mac OS X stores this information in the kernel and uses it when the computer is put to sleep to determine all memory areas whose content needs to be saved onto the hard disk. This information is stored in the `MemoryMap` data structure, whose address can be determined via FireWire by reading `PE_state` and `boot_args` kernel data structures as illustrated in Fig. 2.

`PE_state` is a global data structure in Mac OS X kernel that contains data of the Platform Expert object. It is defined in the file `pexpert/pexpert/pexpert.h` in the kernel source code tree:

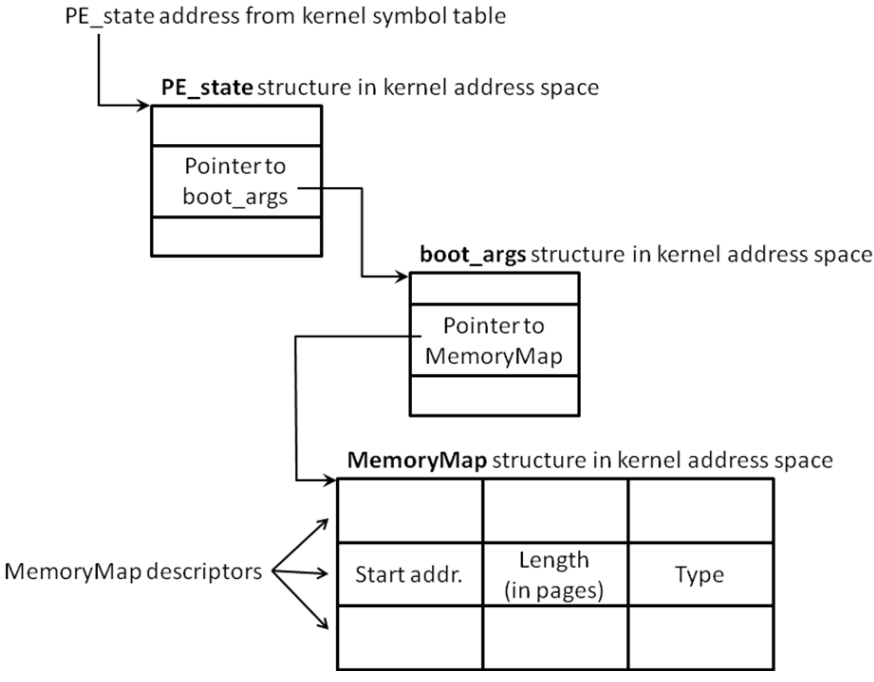

Fig. 2. Accessing Mac OS X memory map

```
typedef struct PE_state {
          boolean_t initialized;
          PE_Video video;
          void  *deviceTreeHead;
          void  *bootArgs;
} PE_state_t;
```

Among other things, the `PE_state` structure contains a pointer called `bootArgs`, which points to the boot arguments passed to the kernel by the firmware at the boot time.

The boot arguments are organized into a struct called `boot_args`, which is defined in the file `pexpert/pexpert/i386/boot.h`:

```
typedef struct boot_args {
  uint16_t    Revision;
  uint16_t    Version;
  char        CommandLine[BOOT_LINE_LENGTH];
  uint32_t    MemoryMap;
  uint32_t    MemoryMapSize;
  uint32_t    MemoryMapDescriptorSize;
  uint32_t    MemoryMapDescriptorVersion;
…
} boot_args;
```

One of boot_args fields, MemoryMap, is a pointer to the memory map of the computer. The memory map is an array of memory map descriptors, where each descriptor defines the location, type, and size of a particular area of memory. A memory map descriptor has the following structure (defined in pexpert/pexpert/i386/boot.h):

```
typedef struct EfiMemoryRange {
  uint32_t Type;
  uint32_t Pad;
  uint64_t PhysicalStart;
  uint64_t VirtualStart;
  uint64_t NumberOfPages;
  uint64_t Attribute;
} EfiMemoryRange;
```

The Type field determines the purpose of the area of memory (e.g. normal RAM vs. memory mapped I/O registers). The PhysicalStart field contains the physical address of where the memory area begins. The NumberOfPages determines the length of the memory area in pages.

The address of PE_state data structure can be determined from the kernel symbol table. The offsets to bootArgs and MemoryMap pointers can be calculated by adding up sizes of the preceding fields in the PE_state and boot_args data structures respectively.

Once the RAM imaging application has determined the version of Mac OS X in the target computer, it must use the address of PE_state and offsets to bootArgs and MemoryMap pointers for that version of Mac OS X. To determine which areas of memory are safe for reading, the RAM imaging application should perform a sweep of the memory map array and identify all memory areas corresponding to some form of RAM. The content of the identified RAM areas should then be acquired via FireWire.

After the RAM image is created, it needs to be analyzed. This paper describes a work in progress whose aim is the recovery of unsaved instant messenger conversations from the RAM of the target computer. Some initial results for recovery of AOL Instant Messenger (AIM) conversation fragments have been obtained, and they are described in the next section.

3 Extracting Fragments of Open AIM Conversations

3.1 AOL Instant Messenger Background

AOL Instant Messenger (AIM) is a peer-to-peer instant messaging service similar to many other “chat” type programs. Below are some of the features and services AIM provides:

- A user can have up to 1000 buddies added to his/her contact list.
- Ability to add, delete or block any buddies in buddy list.
- Viewing buddy information and knowing his/her details.
- A user can also send typed text messages to many buddies at a time.

- The ability to share a user's status, files and pictures.
- The ability to use Text (SMS). This feature sends instant messages to the recipient's cell phone.
- AOL provides a free email account.
- AIM toolbar provides the ability to see the status of AOL mail, AOL search functions, and AIM messaging services as a toolbar in your browser.
- In addition AIM is accessible on Windows Mobile Smartphone, iPhone and iPod touch [3].

Due to its availability, features, and ease of use AIM has become very popular.

By default, the Mac OS X version of AIM does not automatically save chat logs. This is a hindrance from an investigative point of view. To obtain the content of an AIM conversation an investigator needs to extract it from RAM. Section 2 covers this.

3.2 AOL Instant Messenger Chat Logs Formats in the RAM

The research described in this paper focused on AIM version 1.5.127. To determine formats of conversation data as stored in RAM, a test conversation was conducted. The content of the virtual memory of the AIM process was then dumped using gcore-1.3 [4], and the resulting virtual memory dump was then examined using hexadecimal editor 0xED [11]. As a result of this experimentation, several possible AIM message formats were discovered. These formats and the recovery of chat logs are described in the following sections.

Format A

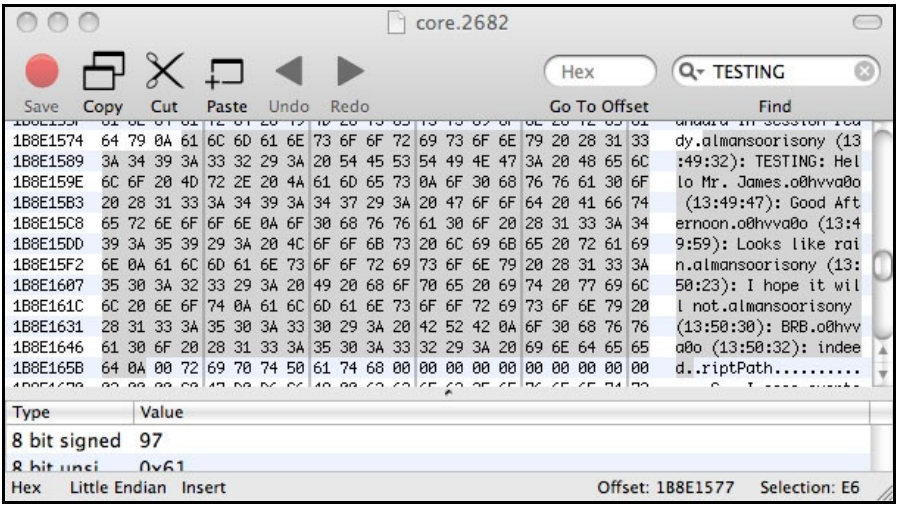

Fig. 3. AIM chat log format A

By searching for known conversation fragments, it was possible to locate conversation data objects in RAM (Fig. 3). From the AIM process memory dump, it can be seen that AIM chat messages have the following format. It contains the user name of the sender, the time when the message was sent, and the text of the message:

```
<USERNAME> (HH:MM:SS): <MESSAGE>
```

The username can only be 3-16 combined letters and numbers, but it should always start with an upper or lower case letter. Given below is a regular expression written in Perl that matches the above message format:

```
m/([a-zA-Z]{1}[a-zA-Z0-
9]{2,15})\s\((\d\d\:\d\d:\d\d)\):\s(.*\s)/
```

Format B. However, after further research it was found that the RAM sometimes contains a different format for the AIM messages, an example of which is shown in Fig. 4.

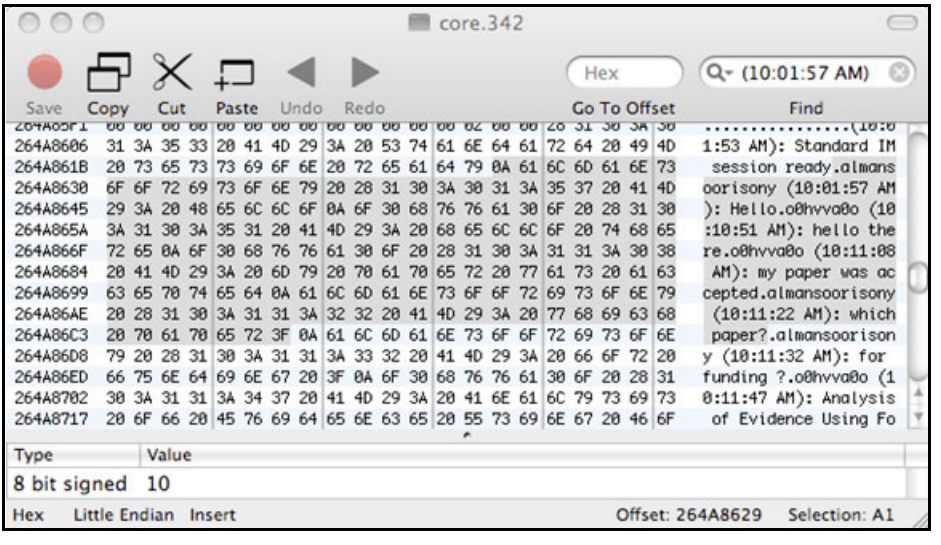

Fig. 4. AIM chat log format B

From Fig. 4 the messages format is:

```
<USERNAME> (HH:MM:SS PM/AM): <MESSAGE>
```

In Perl matching the above chat message format would be the following expression:

```
m/([a-zA-Z]{1}[a-zA-Z0-
9]{2,15})\s(\(\d\d\:\d\d:\d\d\s[A-Z]{2}\):)\s(.*\s)/
```

Format C. Another format for AIM chat logs was also found that has the following format:

```
<USERNAME>: (HH:MM:SS) <MESSAGE>
```

To extract this chat message format the following regular expression in Perl can be used:

```
m/([a-zA-Z]{1}[a-zA-Z0-
9]{2,15}:)\s(\(\d\d\:\d\d:\d\d\))(.*\s)/
```

See Fig. 5 for the above AIM chat log format.

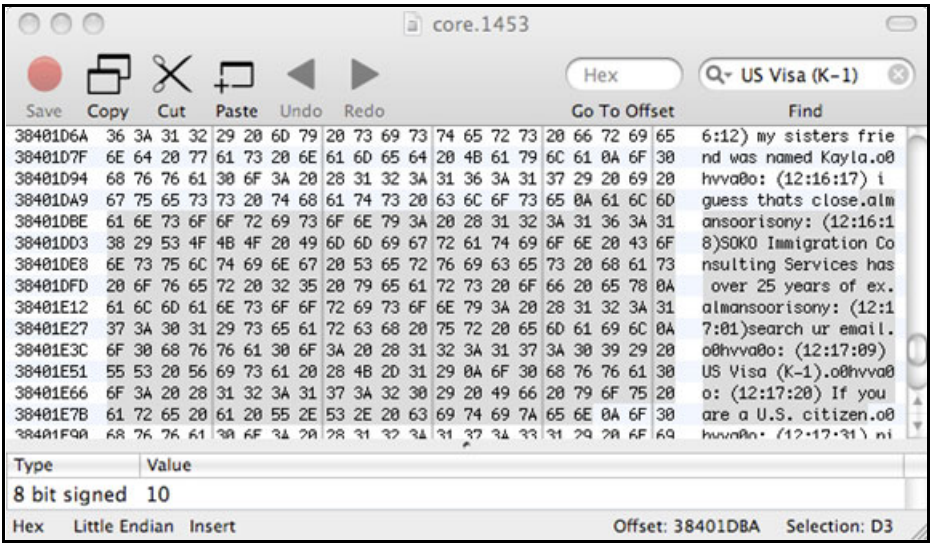

Fig. 5. AIM chat log format C

Format D. From further research, another AIM message format was also found. It is shown in Figure 6.

\\df2 \ almansoorisony

\\bs1 \\bf2

\\bs3 \\bf2 \ Good evening my friend

Fig. 6. AIM chat log format D

The name of the sender (almansoorisony) and the text of the message (Good evening my friend) can be extracted using, for example, the following Perl code:

```
if (m/\\[a-z0-9]{3}\s([a-zA-Z]{1}[a-zA-Z0-9]{2,15}:).*/){
   $username = $1;
   my $firstline = <FILE>;
   $secondline = <FILE>;       #Skip \\bs1 \\bf2
   $secondline =~ m/[^\s]*\s(.*)/;
   $secondline = $1;
   $conversation2 = "$username $secondline";
}
```

The above code will extract the username, assign it to the variable `$username`, and then extract the conversation fragments from the following two lines read from the memory dump file according to the format displayed in Fig. 6.

4 Experimental Results

The techniques described in Sections 2 and 3 have been implemented in a software tool called Goldfish [9]. The tool has been tested on three distinct Apple Mac computers, which were using one of the two different versions of Mac OS X (Darwin Kernel Version 10.0.0 and Darwin Kernel Version 9.8.0). Results have been mixed; in all experiments the tool successfully performed RAM imaging over FireWire without crashing the target computer. In most experiments AIM messages of the ongoing AIM conversations were successfully recovered. However, in some experiments, AIM message recovery did not recover any messages. The reasons for this failure to recover messages are being investigated.

5 Future Work

The research described in this paper is a work in progress. The basic method for extracting RAM content from Apple Mac computers over FireWire has been developed, and the initial steps in the analysis of the RAM dump were performed.

Future work will have to focus on further analysis of the RAM dump. In order to go beyond the simple data carving using regular expressions, it will be necessary to reconstruct the virtual memory spaces of the individual user processes. A method for doing so has been proposed in [6]. Additional difficulty is caused by the inconsistencies in the RAM dump. RAM imaging over FireWire is a lengthy process that happens concurrently with the normal operation of the target computer. As a result, the content of some memory pages may have changed during the imaging process, and the data collected at the beginning of the imaging may contradict the data collected at the end of the imaging. The future research will have to come up with a way of identifying and working around such inconsistencies.

References

1. Boileau, A.: Hit by Bus: Physical Access Attacks with Firewire, `http://www.security-assessment.com/nz/publications/security_publications.html`
2. Boileau, A.: Releases: Winlockpwd, Pythonraw1394-1.0.tar.gz, `http://www.storm.net.nz/projects/16`
3. Wikipedia Article: AOL Instant Messenger, `http://en.wikipedia.org/wiki/AOL_Instant_Messenger`
4. Singh, A.: Process Photography on Mac OS X (Handcrafting Process Core Dumps), `http://www.osxbook.com/book/bonus/chapter8/core`
5. Apple Open Source Connection, `http://opensource.apple.com/`

6. Suiche, M.: Advanced Mac OS X memory analysis, Presentation at BlackHat Briefing Washington DC (February 2010), http://www.blackhat.com/presentations/bh-dc-10/Suiche_Matthieu/Blackhat-DC-2010-Advanced-Mac-OS-X-Physical-Memory-Analysis-wp.pdf
7. Apple Developer Connection, http://developer.apple.com/
8. Unified Extendible Firmware Interface Specifications, http://www.uefi.org/specs
9. Goldfish tool, http://cci.ucd.ie/goldfish
10. Becher, M., Dornseif, M., Klein, C.N.: 0wn3d by an iPod. In: Proceedings of PacSec 2004 Applied Security Conference, Tokyo (2004), http://pacsec.jp/psj04/psj04-dornseif-e.ppt
11. 0xED hexadecimal editor, http://www.suavetech.com/0xed/0xed.html