

Forensic Investigation into Mac OS X Volatile Memory

Tom Coppock and Diane Gan

CSAFE Centre, School of Computing & Mathematical Sciences, University of Greenwich, UK
D.Gan@gre.ac.uk

Abstract. An important area for forensic investigations is live memory analysis captured from a running machine. The RAM may provide an in depth picture of the system when it was seized which could reveal many vital pieces of evidence otherwise unobtainable on the computer hard disk. Research in this area is relatively low on all platforms, but especially for Mac OS X. The aim of this work was to investigate volatile memory analysis for a Mac and to develop a tool, called VolaGUI, to assist forensic examiners when analyzing volatile memory.

Keywords. Mac OS X, forensics, RAM Dump, VolaGUI, Volafox, mach-o, linear

1 Introduction

Digital evidence is increasing in importance in court cases. It consists of traces left on digital devices from which one can infer information about the actions performed on those digital devices. Hidden processes, the increasing presence malicious code/rootkits and the increasing use of encryption keys means that it is sometimes deemed unacceptable for a digital forensic examination of a computer to consist of only the hard drive. Many processes that cannot be viewed via the hard drive can be viewed, in plain text, from a RAM dump. The fact, also, is that some data that is encrypted can be viewed in human readable format from RAM, which is why this resource should never be overlooked whilst conducting the forensic examination of a computer. Information found in RAM is very substantial and forensic examiners should make it a protocol to include this in their normal examination. However, it can also be stated that structured volatile memory analysis for OS X is not simple. The ACPO guidelines state that any tool used to acquire a RAM dump of a machine (Mac or Windows based) must be lightweight, as it cannot leave a footprint on the target machine [1].

This paper presents a brief overview of three currently available live memory analysis tools for the Mac OS X and then discusses the application developed during the course of this work, called VolaGUI. The objective was to extend the existing Volafox application, to provide an intuitive graphical user interface and to format the output of the evidential information found in volatile memory so as to make it a useful tool for forensic investigators. The results from the tool show the product in use, and the conclusion will critically evaluate the product. This paper also discusses potential further development.

2 An Overview of Live Memory Analysis Tools

Three tools are available to analyze volatile memory from Macs and these are Volafox, Goldfish and the Volatility framework, discussed in this section. They are of varying complexity and challenging to use.

- Volafox is an open source (GNU GPL) command line based tool, based upon research by Suiche (2010) [2] and the Volatility memory analysis framework. It is written in Python 2.5, making it cross platform. It encompasses the idea of symbol table re-generation. Volafox is built on code heavily recycled from the Volatility framework [3]. Previous revisions of Volafox required the mach_kernel image to be present so that the symbol table can be built. This was deemed inefficient [4], as it placed heavy reliance on the kernel image itself. Later revisions provided an overlay generator for the RAM dump so that successful symbol generation was possible. Using Volafox requires a script vol.py to be run, followed by the memory dump file name and then the selected options. Volafox is one of the better-documented analysis programs [4], [5], which discusses the incremental development of the core Volafox implementation.
- Goldfish is a closed source program available to Law enforcement only. As such very little documentation and research is available. It provides a facility in which the examiner can obtain a RAM image via a firewire connection. Once the image is acquired, a command line interface can be used to recover the system password and AIM (AOL Instant Messenger chat fragments) [6].
- Volatility is an open source framework written in Python and implemented under the GNU General Public License. It is used to analyze memory from machines running Windows, Linux and Mac OS X [7]. Volatility benefits from a large developer base and as such frequent features and updates are delivered to the forensic community. However, the Mac analysis feature has only been added in version 2.3 [8] and is still undergoing continual development. Volatility can be installed in various forms, depending on the native OS. There is a standalone executable version available for Windows but for Linux and Unix system source based installation is the only option. Volatility comes with all of the necessary tools to create a profile. However, one issue that has been identified is that the Mac section suffers from a lack of development and research, so this is not a good candidate.

Although the above tools can offer a simple way of finding data, they are not without their limitations. Although the command line syntax is relatively simple to use, for the inexperienced user this could be challenging. In an industry where an increasing number of Police officers are being trained to examine digital devices and who are not trained IT experts, the need for simpler and more intuitive tools is becoming more important. Also, these tools were not specifically designed as forensic tools as they offer no facilities for reporting. This means that the examiner will have to spend time formatting data to put into their report, thus increasing the time spent in the overall examination process.

There are generally two extremes when it comes to using these tools. You are either faced with a program with very beneficial features (from a forensic point of view), which is difficult to use, or a program, which may be considered cumbersome in examination but is easy to use. It has become apparent that no open source programs offer a middle ground of these desirable features. Hence the concept to make the Volafox program more accessible and easier to use for the forensic community.

3 VolaGUI

This project set out to develop the open source Volafox toolkit to make it easier and more intuitive to use. This meant integrating code into the existing product without losing any functionality or compromising the forensic integrity of the data. The tool VolaGUI was written in Python and Flask and used HTML and CSS to format the output.

3.1 Requirements

The finished product, VolaGUI, should be integrated into Volafox without any loss of functionality. It should have an intuitive interface. The main reason being that many Police Officers, who may use the software, may not want to use the Volafox command line interface. The finished product should be cross platform, to ensure maximum portability and usage across all operating systems and should not use excessive computer resources. It should be able to act transparently, causing no noticeable impact on performance.

There are a number of assumptions that must be made. Current methods dump RAM in different ways, namely mach-o format, Linear or raw format [9]. The example RAM dumps were obtained from the program Mac Memory Reader which produces RAM dumps in the native mach-o format. It was also assumed that the RAM dump had been obtained in a forensically sound way that would make it admissible in a court of law. It was also assumed that this would be the format of the RAM dumps to be analyzed by VolaGUI.

3.2 Functionality

Volafox allows RAM dumps to be in several different formats. The format most commonly used is RAM acquired using the tool MacMemoryReader, which dumps RAM in a mach-o format. In the case that the RAM is in mach-o format, the RAM dump needs to be flattened to perform the examination. This is done using the `flatten.py` python script, which is a utility that comes with Volafox. The script converts the dump to a linear format so Volafox can correctly read it. When the script `vol.py` is executed, it checks if the RAM dump is in the correct format. Depending on which type of image is used (mach-o or linear), the RAM dump file `machaddrspace.py` or `addrspace.py` will be called, respectively. These files are responsible for file operations, such as specifying address spaces and ranges, on the varying types of RAM

dumps. The open source code `addrspace.py` comes from the Volatility Framework and `machaddrspace.py` was written specifically for Volafox to enable Mac OS X RAM dumps in mach-o to be analyzed. The overlay used in the command arguments is then opened and carries out physical to virtual address translations. With a 32-bit architecture it uses the `ia32_pml4.py` file and the 64-bit architecture uses the `x86.py` file. The next step is to branch to a specific Volafox method. Each method accepts a kernel symbol and returns a string matrix of results [4].

Flask was used as the web development platform as it integrates well into python. To refer to a path in Flask, the `render_template()` method is used and a html file is passed as an argument. Flask expects the templates to be stored in the `templates` folder in the project root directory.

The function Hijack STDOUT enables the web application to display the output of the underlying Volafox program in a nicely formatted web page. Hijack STDOUT takes use of the module `StringIO`, which allows strings to be read in as files. The process carried out is the following:

- Sets a variable called `Temp`
- Sets the standard output (`sys.stdout`) to `Temp`
- Loads the RAM dump and seeks the beginning of the file

Once the RAM dump is loaded, the user can select an analysis plugin to be applied to retrieve the required information. As the RAM dump is located in the variable `Temp` during the analysis, access times within the evidence are not compromised as the process, (`StringIO`) uses CPU and memory and not the hard drive.

3.3 Testing VolaGUI

The application has been tested and the results are presented here. Fig. 1 shows the home page for VolaGUI. As can be seen the style is simple and feedback is given to the user on the first page so that they stay informed throughout the examination process.



Fig. 1. VolaGUI home page

Fig. 2 shows the page where the user will load the RAM dump into VolaGUI. There are only certain places a user can click so the risk of user error is minimized. The user is then notified if the RAM dump has been successfully loaded and the menu is loaded to enable the user to proceed to the examination. If an error occurs, the user

will be notified.

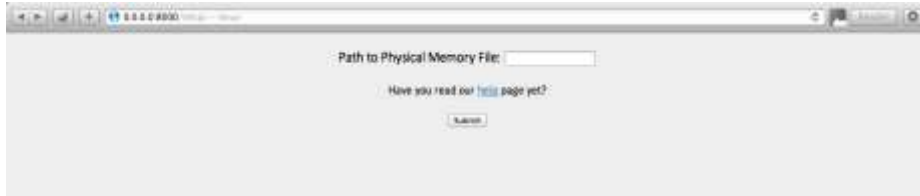


Fig. 2. VolaGUI setup page

The menu gives the user the choice to select an analysis module to retrieve data from the RAM dump, see Fig. 3. The data returned will be identical to the original Volafox program due to the Hijack STDOUT function, which grabs the output from the program, and formats it to a webpage

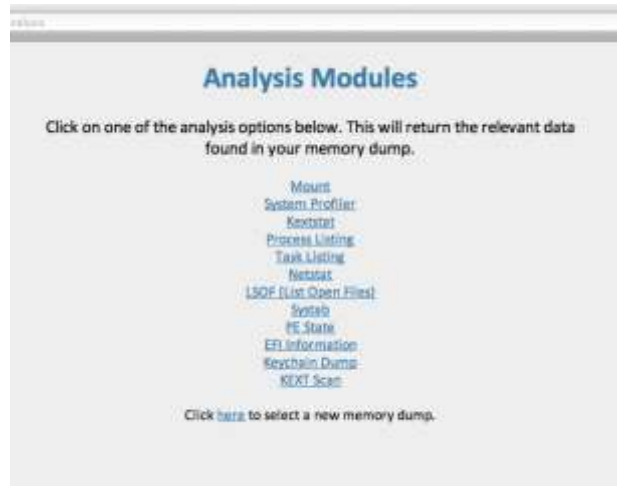


Fig. 3. Analysis modules page

The first option is Mount, which returns information regarding the file systems that were mounted on the machine at the time computer was seized. For example, if a USB stick has been seized along with the computer, it can be determined if it had ever been connected to that machine. This option also allows the investigator to view separate partitions. The RAM dump shown in Fig. 4 is an example of the output given to the user from VolaGUI. It shows that a Bootcamp partition, related to the Windows OS, was present. This information can be very useful to the examiner and could prompt investigation into the newly discovered partition.

```

[+] Mount List
NEXT ENTRY FS TYPE MOUNT ON NAME MOUNT FROM NAME
0xFFFFFFFF800A88D188 HFS /dev/disk1
0xFFFFFFFF800A88C7A0 DEVFS /dev/devfs
0xFFFFFFFF800A88BDB8 NTFS /Volumes/BOOTCAMP /dev/disk0s4
0xFFFFFFFF800A88B3D0 AUTOFS /net map -hosts
0xFFFFFFFF800A88A9E8 AUTOFS /home map auto_home
0xFFFFFFFF800A88A000 NFS /Volumes/MobileBackups localhost/xp53Ha4a2bhsG8rclgrdgE
0x00000000 HFS /Volumes/Recovery HD /dev/disk0s3

```

Fig. 4. VolaGUI output from Mount analysis

The module Tasks returns a list of tasks running on the machine, see Fig 5. for a sample section. A useful feature of this is that the user's name is also listed which can be particularly useful if, for example, the case related to a hacking attempt.

```

[+] Task Count at Kernel Symbol: 114
[+] Task List Count at Queue: 114
[+] Process List Count: 114
[+] Linked task list
TASK CNT OFFSET(P) REF_CNT Active Halt VM_MAP(V) PID PROCESS USERNAME
0 0x05216C40 85 1 0 0xFFFFFFFF80025BBE80 0 kernel_task
1 0x052168B8 5 1 0 0xFFFFFFFF80025BB2B8 1 launchd _softwareupdate
2 0x052161A8 3 1 0 0xFFFFFFFF800B041DE8 14 kextd root
3 0x05217A98 5 1 0 0xFFFFFFFF80025BB000 15 UserEventAgent root
4 0x05217E20 4 1 0 0xFFFFFFFF800B0413F0 16 mDNSResponder _mdnsresponder
5 0x05217710 11 1 0 0xFFFFFFFF800B041D00 17 opendirectoryd _mdnsresponder
6 0x05217388 5 1 0 0xFFFFFFFF800B041C18 18 notifyd _mdnsresponder
7 0x05217000 3 1 0 0xFFFFFFFF800B041B30 19 diskarbitrationd _mdnsresponder
8 0x1EAD5C40 8 1 0 0xFFFFFFFF800B041A48 20 configd _mdnsresponder
9 0x1EAD58B8 3 1 0 0xFFFFFFFF800B041960 21 feventd _mdnsresponder

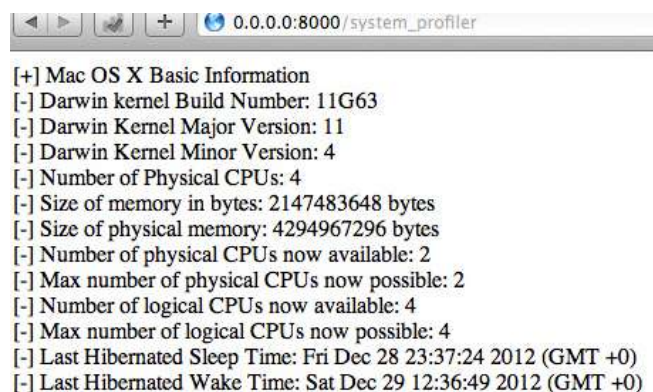
```

Fig. 5. VolaGUI output of Tasks analysis

The module System_profiler returns useful information about kernel versions, CPU information, the last time the system was in sleep mode and the wake-up time, see Fig. 6.

In cases where the user is accused of downloading indecent images and claims that a Trojan was responsible, Systab analysis can identify malware by detecting system hooking. Other functions include Kextstat, which determines if the currently loaded kernel is a genuine copy of the OS X. Illegal copies may be unstable and unpredictable which can cause problems for the investigator. Similar to Kextstat is Kextscan, which can help to determine exactly what kernel modules were in use at the time of seizure. Netstat is included to determine open ports and the services running on them. EFI Info returns information regarding the Extensible Firmware Interface. This can be used to determine hardware information, boot service calls and pre-boot applications. The Keychain Dump option will provide master key candidates from the central store

where all security information, such as passwords and certificates, are located. Also included are PE State which shows the boot process and LSOF (list open files) [4] which will show the examiner exactly what was running at the time of seizure.



```
[+] Mac OS X Basic Information
[-] Darwin kernel Build Number: 11G63
[-] Darwin Kernel Major Version: 11
[-] Darwin Kernel Minor Version: 4
[-] Number of Physical CPUs: 4
[-] Size of memory in bytes: 2147483648 bytes
[-] Size of physical memory: 4294967296 bytes
[-] Number of physical CPUs now available: 2
[-] Max number of physical CPUs now possible: 2
[-] Number of logical CPUs now available: 4
[-] Max number of logical CPUs now possible: 4
[-] Last Hibernated Sleep Time: Fri Dec 28 23:37:24 2012 (GMT +0)
[-] Last Hibernated Wake Time: Sat Dec 29 12:36:49 2012 (GMT +0)
```

Fig. 6. VolaGUI output from System_profiler

4 Future Developments

There is still more work to be done to develop the tool. One useful addition would be to add the functions `flatten.py` and `overlay_generator.py` to VolaGUI which will enable users to complete a full examination without having to touch any command line interfaces.

A new analysis module should be included which will extract WAN and LAN data, giving the information regarding specifically connected networks and previously connected networks. Currently there is a lack of modules to determine network information from the running machine. At the moment, there is only one module (`netstat`) which does this.

One of the problems with data stored in volatile memory is that there is a wealth of data available, but it can be very cluttered and disorganized. Depending on the size of the RAM dump, it can be very difficult to extract all relevant data from the dump. As such, it would be wise to carry out research into the field itself, as opposed to being tool specific, so that there can be an organized and structured method of obtaining all of the relevant data from the machine.

Other areas of potential future development follow a less conventional route, yet may prove useful to forensic examiners. Recently, a Python compiler for iOS [10] has been developed. Using knowledge of the architecture of Volafox and VolaGUI, this could be adapted to examine volatile memory for iOS devices. This has the potential to be useful, as these devices are usually examined “live” and the examiner is very limited as to what they can do. Volatile memory analysis for these devices can have

the same effect as they would for OS X boxes in that they can provide information to the examiner not often obtainable by any other method.

Another area of development is volatile memory analysis for Hackintosh machines. A Hackintosh is a computer running Mac OS X but is running on custom hardware. This is done for a number of reasons, such as performance and cost. These Hackintosh machines have the same architecture as standard OS X machines, but can be considered temperamental, due to the nature of installation. A bespoke tool that can analyze volatile memory from these machines would be invaluable to examiners.

5 Conclusion

The product can be considered a success, as the aim was to create a tool which was easy to use and could be used to analyse volatile memory from computers running Mac OS X in a straightforward and easy manner. Python and Flask seemed a very wise choice for the development. Flask integrates very well with Python code, and has properties allowing “Rapid Web Application Development”.

Both Volafox and VolaGUI were given to a number of users, who were not familiar with either forensic tools or command line environments, to determine if the tool was easy to use. All users were able to use VolaGUI and none could use Volafox. This is a clear indication that the tool was easy to use and intuitive. The conclusion from this is that volatile memory analysis is fast becoming more accessible to more forensic examiners, meaning that they can now incorporate volatile memory into their examinations seamlessly.

References

1. Good Practice Guide for Computer-Based Electronic Evidence, ACPO (Association of Police Chief Officers) Pp. 7, (2011)
2. Suiche, M., Mac OS X Physical Memory Analysis, BlackHat Security Conference, Washington DC, February 2010
3. Case, A., Mac Memory Forensics with Volatility, (2011) <http://computer-forensics.sans.org/summit-archives/2012/mac-memory-analysis-with-volatility.pdf>
4. Hay, A. F., Forensic Memory Analysis of Apple OS X, Thesis, Air Force Institute of Technology, Wright-Patterson Air Force Base, Ohio, June 2012
5. Leat, C., Forensic Analysis of Mac OS X memory, Master's thesis, University of Westminster, (2011),
6. Goldfish, Digital Forensics Investigation Research Laboratory, http://digitalfire.ucd.ie/?page_id=430, (2012)
7. Volatility – an Advanced Memory Forensics Framework, <http://code.google.com/p/volatility>
8. Volatility 2.3 Release Notes, Volatility Framework, <http://code.google.com/p/volatility/wiki/Release23>, (2013)
9. Volafox - Mac OS X & BSD Memory Analysis Toolkit, <http://code.google.com/p/volafox>, (2012)
10. Hosmer, J., Python for iOS, <http://pythonforios.com/>, (2013)