Chapter 1

A COMPILED MEMORY ANALYSIS TOOL

James Okolica and Gilbert L. Peterson

Abstract Performing a forensic investigation on computer memory has increased in importance for forensic investigators. Volatile memory analysis can provide indicators on what to search for on the hard drive, passwords to encrypted hard drives, and legal support against a defense claiming innocence and that the computer was infected with malware. Historically, investigators would perform a live response consisting of executing several utilities to retrieve this information, by using a single tool to capture the memory and then analyze it is both more efficient and has less impact on the system state. This paper discusses several methods for dumping memory and presents a new self-contained memory analysis tool, CMAT, for extracting forensic information from a memory dump into a format suitable for further analysis. Results from CMAT compared with several utilities commonly used in a live response, demonstrates that CMAT provides comparable information and identifies a malware process the utilities miss.

Keywords: live response, memory analysis, rootkit detection

1. Introduction

When a large organization detects a cyber security incident, performing the NIJ recommended disconnection and imaging of the hard drive process to gather forensic evidence [14] is not always possible, especially on a critical server. As an example, during Amazons outage in June 2008, CNET reported that Amazon lost $29,000 in revenue for each minute it was down [6]. In addition, there have been several legal cases recently where defendants were either found innocent or guilty based on the live response information retrieved at the time of the incident [2]. For both of these situations, tools that can take a quick snapshot of the affected computers while minimizing impact on the business and on the hard drive evidence are vital for digital forensics. Since many of the

items of interest in a live response are maintained in the memory of the computer rather than the hard drive, a minimally intrusive live response would be to strictly analyze a live memory capture.

There are three steps needed to develop a minimally intrusive standalone forensic tool for memory: 1) a front-end memory dump routine that directly accesses physical memory, 2) a memory analysis tool that extracts environment and activity information from the memory dump and 3) a synthesis tool that correlates the environment information to provide a human-understandable narrative of what was occurring on the machine. This paper focuses on the second step: developing a compiled memory analysis tool (CMAT) that extracts environment and activity information from a memory dump. CMAT takes an existing memory dump and parses through it to find active, inactive and hidden processes as well as system registry information. It then compiles live response forensic information from these processes and registry files and assembles it into a format suitable for data correlation. CAMT is tested against several utilities that extract forensic information from a live system. Results show that CMAT provides the same information that these utilities provide and, additionally, uncovers a hidden process that these tools miss.

2. Background

There are two reasons to capture forensic information to mitigate damage and to prosecute the perpetrators. Performing a live response allows the capture of forensic information that disappears after the computer is turned off. The benefits of a live response include: gathering clues to better focus a search of hard drives [24], retrieving decryption/encryption keys for encrypted hard drives [24], and defeating the Trojan horse defense of I didnt do it, it was the malware installed on my machine [2]. The live response information investigators seek include:

- system date and time,
- logged on users and their authorization credentials,
- network information, connections and status,
- process information, memory and process-to-port mappings,
- clipboard contents,
- command history,
- services, driver information,

- open files and registry keys as well as hard disk images [3, 5, 12, 24].

With the exception of the hard disk images, all of the information resides in memory.

One of the weaknesses of live response is the concern that the evidence capture modifies the machine. Even the act of performing a memory capture modifies the machine under investigation. There have been several techniques proposed to address this issue [1, 4, 10, 22]. What all these tools have in common is capturing all of memory rather than capturing specific pieces of forensic information. This allows not only answering the immediate forensic questions but also provides a means of answering additional questions later in the investigation. These alternatives include hardware-based memory acquisition [1, 4], virtual machines, hibernation files [22], and operating system patches [10]. Hardware-based memory acquisition circumvents reliance on the operating system by using a piece of hardware that interacts directly with memory. These techniques include custom hardware [4] and using the firewire interface [1] for direct memory access. However, these two techniques suffer from the Northbridge exploit [17]. Since all peripheral devices use the North bridge to interface with the CPU and main memory, malware can be loaded onto the North bridge to subvert memory access by a peripheral while leaving the CPU and operating system untouched. The result would be a piece of malware that could remain undetected.

One of the most promising ways to avoid the Northbridge exploit is to constrain the memory capture techniques to the CPU and operating system. Virtual machines provide an easy method for dumping memory. When an incident occurs, a forensic investigator takes control of the actual machine, suspends the virtual machine, and then dumps the memory in the virtual machine. There are two significant issues with this approach. The first is the same as the custom hardware solution: enacting such a policy within an organization of any significant size and geographic distribution incurs high costs in time and money. The second is that any software virtualization will slow the user perceived responsiveness of the computer since every command goes through an additional level of abstraction. Hibernation files are another memory capture method that rely on the operating system but do not result in a slow down in user perceived responsiveness [22]. The process of seizing and analyzing hibernation files requires placing the system in hibernation mode, copying the hibernation file off of the system, and then extracting the memory contents from the hibernation file. There are two significant problems with this method. The first is that while this

function is enabled on many laptops, it is disabled by default on most desktops. Although this can be overcome by configuration, the configuration must be performed ahead of time and it must be protected against disabling of the function. The second issue with hibernation files is the lack of a published description of the hibernation file format and that not everything in memory is stored.

Another proposed method for acquiring a pristine copy of memory was made by Libster and Kornblum [10]. They propose modifying the operating system kernel so that a user defined function key would cause the operating system to immediately suspend, generate a memory dump that it (possibly) sends over a network to a secure machine, and then resume. The greatest weakness of such a method is the need to ensure its deployment across all of an enterprises disparate systems prior to an incident.

While an operating system patch seems an ideal solution if pre-installed, in those cases where nothing has occurred prior to an incident, the only option is to initiate a new process to dump the systems memory. To minimize the impact, the application should limit (or totally avoid) application program interfaces (APIs) that interface with the operating system and the use of a graphical user interface (GUI). Unfortunately, while many available tools [13, 23] are run from the command line, most use operating system APIs to dump memory. The one exception appears to be Memoryze [11].

There are several memory analysis tools that extract different amounts and types of live response data from a memory dump. Volatile Systems Volatility tool [25] extracts the most in parsing through a memory dump. Volatility provides tools for analyzing a memory dump and the ability for plug-ins to be added. While the framework itself only extracts date and time, running processes, open ports, process to port mappings, strings to process mappings, process to file mappings, and process dlls, and open connections [25]; several developers have written additional plug-ins enhancing the functionality of Volatility. The most significant of these additions is the ability to extract registry information. The greatest limitation of Volatility is that it requires Python to run. While this might not seem like a large drawback since Python is available for public download, it does require the computer analyzing the memory dump to have Python loaded. In most cases, where analysis occurs on a separate forensic machine, this is not an issue. However, in those instances where analysis needs to occur locally, a compiled memory parser would be preferable. Unfortunately, there are no comprehensive open source memory analysis tools for Windows XP written in compiled languages like C/C++. While Chris Betz has developed a tool [21] that finds pro-

cesses and threads in memory and extracts some information, his tool is limited to Windows 2000 and does not include analysis of the registry files, network activity, or process creators. The PTFinder [19] tool also searches through memory for processes and threads, and handles memory captures from Windows XP. However, like Betzs work, it does not include analysis of the registry files [7], network activity, or process creators [9].

3. CMAT: The Compiled Memory Analysis Tool

The compiled memory analysis tool (CMAT) is a C++ command line utility that runs on both Linux and Windows operating systems and parses a Windows XP memory dump to provide information on user accounts, the Windows Registry, and running processes. CMAT provides system, process, registry, and user information in a standalone tool that runs without API calls or high level language interpreters. CMAT conducts two passes. In the first pass, CMAT searches the memory dump file for processes, threads and registry hives. In the second pass, CMAT takes the process, thread and registry entries it has found and produces output in a format similar to that produced by common live response tools.

The Windows kernel has data structures for the data required for the operating system to function. These data structures are useful for extracting the live response information. For example, processes are stored in memory as _EPROCESS structures. By looking for strings formatted as a _DISPATCH_HEADER of a _KPROCESS, potential process structures can be found. For Windows XP SP2, this corresponds to a _DISPATCH_HEADER of 0x03 and a _KPROCESS size of 0x1b. Once likely processes are found, they are double checked by ensuring the page directory table base is a valid virtual memory location [15, 18]. If the page directory table does not point to a valid location, then what has been found is executable or data of a process and not a process header. This method finds all processes, including normal active processes, hidden active processes (including processes hidden by a rootkit like FUTo [20]), defunct processes, and processes from previous boots that have not been removed from memory; in addition, since the _EPROCESS structure also contains a doubly linked list of all active processes, any processes missed during the string search can still be found by traversing the linked list. The scan of the linked list also assists in identifying found process that have been disconnected from the list which is how FUTo hides processes.

CMAT then searches through the memory dump for registry hives. In a manner similar to searching for processes, CMAT looks for _CMHIVE structures with a signature in Windows XP of 0xbee0bee0 [7]. Also, just as processes must have a valid memory address for its page directory table base, registry hives must have a valid memory address for their base block. Finally, CMAT also takes advantage of the doubly linked list of _CMHIVEs to ensure all the registries are found. Once CMAT has found all of the registries, it moves through the default user registry to find a relationship between session IDs and user names. In Windows XP, it finds the ProfileList key under \USER\MICROSOFT\Windows NT\CurrentVersion and then makes a record of each of the session IDs and their associated ProfileImagePaths providing human understandable names for the users that are logged on.

As a part of its processing, CMAT must translate the virtual addresses stored in the Windows kernel data structures into physical memory locations. Windows XP when run on a 32 bit machine has two levels of indirection, achieved by splitting a virtual address into a page directory index, a page table index, and an offset. However, if large pages (4 MB) are used, there is one level of indirection and if physical address extensions are used there are three levels of indirection. While CMAT is able to detect if large pages are being used, it currently requires a parameter to be passed to determine whether physical address extensions are in use. One method to avoid this is to try alternative virtual to physical mappings until the correct one is found [25]. An alternative method would be to find the kernel system variables that store this information. Some promising work in this direction has been done by [8].

Once the process entries and registry hives are found, CMAT either interactively or in batch mode provides forensic information of interest including general system information (the operating system and number of processes), and process information, which includes the user who created the process, the full path of the executable, the command line used to initiate the process, the full paths of the DLLs loaded and the files and registry keys accessed by the process.

4. Testing Methodology

To test CMATs functionality, the output of CMAT is compared with output from SysInternals psinfo, pslist, logonsessions, handles, and listdlls utilities [16]. The system information examined includes operating system version, number of processors, and number of processes. The process information examined includes process creator, files opened, registry keys accessed, and modules loaded. The one type of volatile informa-

tion that is not examined is network information including the ports and sockets opened by processes since at this time CMAT does not provide this functionality. Lastly, a comparison is made of the number of distinct dynamic link libraries used by Mantechs Physical Memory Dump utility compared with the number of distinct dynamic link libraries used by the SysInternals utilities.

The testing environment is a Windows XP SP3 system with 2038 MB of RAM. Several application programs are started on the machine including Internet Explorer, Word, PowerPoint, Visual Studio, Calculator, Kernel Debugger, and two command line shells. One of the command line shells is hidden by the FUTo rootkit [20]. The memory dump is collected using ManTechs Physical Memory Dump Utility version 1.3 and takes 67 seconds.

5. Results

Test results, summarized in Table 1, demonstrate that CMAT provides the same or equivalent information to the information provided by the SysInternal utilities. In addition, CMAT found the process hidden by FUTo while the SysInternal utilities did not. Additionally, the SysInternal utilities failed to associate several processes with the user that started them. In all, except for the inability of CMAT to provide network information, CMAT performance in the tests was exemplary.

System Information Both CMAT and psinfo provide the operating system version, major and minor version, service pack number and build number. They also both provide the number of processors. However, psinfo also provides the speed and type of processor as well as the video driver used. At this time, while available, this information appears to be of limited use in a forensic investigation and so the decision was made not to implement it in CMAT. The one piece of information from psinfo that would be useful for CMAT is whether physical address extension is used (currently it is passed in via parameter). A methodology for overcoming this is discussed in future work.

Process List - The SysInternals utility pslist provides a list of all active processes. When compared with CMAT, there was 97% equality in comparison. Pslist reported 57 processes and CMAT reported 58 processes, 56 of which matched. The program missing from the CMAT results is pslist which was not running when the memory dump occurred. Similarly, the program mdd_1.3 was missing from pslist which makes also sense since the memory was already dumped when pslist was run. The other program missing from the pslist results is the cmd.exe that was hidden by FUTo, showing that a program intentionally hidden

Table 1. CMAT vs. SysInternals Utilities.

Information	Result	Correlation(%)
Operating System Version	Windows XP SP3 v5.1 build 2600	100
Processor Count	2	100
Process Count	56 of 58	97
User IDs	50 of 57	88
Loaded Modules	57 of 57	100
Files	57 of 57	100
Registry Keys	57 of 57	100
DLLs Used	15 (CMAT) versus 48 (SysInternals)	

by malware can still be found by CMAT. While CMAT provides a single process listing that includes the user who created the process, SysInternals uses a second utility, logonsessions, for this purpose. When CMAT and logonsessions are compared, nine processes do not appear anywhere in logonsessions including all of the processes owned by LocalService and NetworkService as well as four processes owned by SystemProfile (including the system and idle processes). Capturing of the system processes is important for detecting if a malicious service has been started on the machine.

Processes Loaded Modules The output from CMAT and SysInternals listdll utility were checked for all processes and matched 100

Processes files and registry keys The list of files and registry keys in general matched between CMAT and handle.exe. The exceptions were files that appear to be temporary files that were opened and closed between the memory dump and the execution of handle.exe. These files had an access of [RWD] in the listing in which they appeared. In addition, while CMAT recognized file handles that were actually device handles, e.g., \Device\KsecDD, CMAT was unable to print the name of the device. Similarly, where SysInternals handle utility summarized the long code for the current user with a succinct HKU, CMAT displayed the entire registry key.

Comparison of Dynamic Link Libraries As previously mentioned, minimizing the number of loaded modules (dlls) used when performing a live response is desirable, as this indicates that less of the hard drive evidence will have been modified. This experiment compared the number of dlls used by Mantechs mdd with the total number of DLLs used by the five SysInternal utilities. To collect the results, the Mantech mdd memory capture tool was executed during the time each of the five SysInternal utilities were running. The list of dlls associated with the SysInternal utilities were extracted from the memory dump us-

ing CMAT. The DLLs used by the five SysInternal utilities were tallied, removing duplicates.

Results underscore that by performing a memory capture rather than a live response has a significant reduction in impact. The memory capture used 15 DLLs while in total the five SysInternal utilities used accessed 48 different DLLs. This is a 69% reduction in system impact for a live response.

6. Conclusions and Future Work

The developed CMAT tool provides more information from a memory dump than executing similar SysInternal utilities during a live response. In addition, by only performing a memory capture, there is a 69% reduction in system impact. However, there are several pieces of desirable forensic information that CMAT is not able to extract. The most noticeable of these is the mapping of ports and sockets to processes, similar to SysInternals graphical user interface portmon utility. The difficulty in retrieving this information is that it is stored in the data section of the TCP/IP module. While for a given version of the Windows operating system, service pack and patch configuration, the port and socket data section is in a specific place, this location has the potential to change whenever a new patch is installed.

A second extension planned for CMAT addresses the issue that it is applicable to Microsoft Windows XP operating systems. For alternative versions of Windows (e.g., Vista, Windows 7), CMAT will likely map the kernel data structures incorrectly. What is needed is the ability to retrieve the operating system data structures dynamically once the specific operating system is parsed from the memory dump. Microsoft has already realized the need for this flexibility and provides these data structures (symbol tables) for download in real time in support of its kernel debugger. The goal is for CMAT to make use of the provided symbol tables to be useful for any version of Windows.

References

[1] A. Boileau, Hit by a Bus: Physical Access Attacks with Firewire *Ruxcon*, http://www.storm.net.nz/static/files/ab_firewire_rux2k6-final.pdf, 2006.

[2] S. Brenner, B. Carrier, and J. Henninger, The Trojan Horse Defense in Cybercrime Cases *Cerias Tech Report 2005-15*, 2005.

[3] B. Carrier, and J. Grand, Hardware-Based Memory Acquisition Procedure for Digital Investigations, *Journal of Digital Investiga-*

tions, 1(1): 50–60, 2004

[4] B. Carrier, *File System Forensic Analysis*, Addison-Wesley Professional, 2005.

[5] H. Carvey, *Windows Forensic Analysis*, Syngress, 2007.

[6] CNET, Amazon suffers U.S. outage on Friday, http://news.cnet.com/8301-10784_3-9962010-7.html, June 6, 2008

[7] B. Dolan-Gavitt, Forensic Analysis of the Windows Registry in Memory, *Proceedings of the 2008 Digital Forensic Research Workshop (DFRWS)*, pp. 26–32, 2008.

[8] B. Dolan-Gavitt, Push the Red Button Finding Kernel Global Variables in Windows, http://moyix.blogspot.com/2008/04/finding-kernel-global-variables-in.html, 2009.

[9] B. Dolan-Gavitt, Push the Red Button Linking Processes to Users, http://moyix.blogspot.com/2008/08/linking-processes-to-users.html, 2009.

[10] E. Libster, and J. Kornblum, A proposal for an integrated memory acquisition mechanism, *ACM SIGOPS Operating Systems Review*, pp. 14–20 ,2008.

[11] Mandiant, Memoryze, http://www.mandiant.com/software/ memoryze.htm, 2009.

[12] C. Prosise, K. Mandia, and M. Pepe, *Incident Response & Computer Forensics, 2ed*, McGraw-Hill/Osborne, 2003.

[13] Mantech, Mantech Memory DD, http://www.mantech.com/msma/mdd.asp, 2009.

[14] NIJ, *Forensic Examination of Digital Evidence: A Guide for Law Enforcement*, US Department of Justice,2004.

[15] M. Russinovich, and D. Solomon, *Microsoft Windows Internals, 4th Edition*, Microsoft Press, 2005.

[16] M. Russinovich, Sysinternals Suite, http://technet.microsoft.com/en-us/sysinternals/bb842062.aspx, 2009.

[17] J. Rutkowska, Beyond the CPU: Defeating Hardware Based RAM Acquisition (part I:AMD Case), *Black Hat DC*, pp ????, 2007.

[18] A. Schuster, Searching for Processes and Threads in Microsoft Windows Memory Dumps, *Proceedings of the 2006 Digital Forensic Research Workshop (DFRWS)*, pp. ?????, 2006.

[19] A. Schuster, PTfinder, http://computer.forensikblog.de/en/2006/03/ptfinder_0_2_00.html, 2009.

[20] P. Silberman, FUTo, http://www.uninformed.org/?v=3&a=7&t=sumry, 2005.

[21] Sourceforge.net, Memparser, http://sourceforge.net/projects/memparser, 2009.

[22] M. Suiche, Sandman Project, *Whitepaper*, 2008.

[23] M. Suiche, Win32dd, http://win32dd.msuiche.net/, 2009.

[24] I. Sutherland, J. Evans, T. Tryfonas, and A. Blyth, Acquiring volatile operating system data tools and techniques, em ACM SIGOPS Operating Systems Review, vol(number), pp. , 2008.

[25] A. Walters and N. Petroni, Volatools: Integrating Volatile Memory Forensics into the Digital Investigation Process, *Blackhat*, 2007.