A Compiled Memory Analysis Tool

James Okolica, Gilbert Peterson

Chapter 14

A COMPILED MEMORY ANALYSIS TOOL

James Okolica and Gilbert Peterson

Abstract The analysis of computer memory is becoming increasingly important in digital forensic investigations. Volatile memory analysis can provide valuable indicators on what to search for on a hard drive, help recover passwords to encrypted hard drives and possibly refute defense claims that criminal activity was the result of a malware infection. Historically, digital forensic investigators have performed live response by executing multiple utilities. However, using a single tool to capture and analyze computer memory is more efficient and has less impact on the system state (potential evidence). This paper describes CMAT, a self-contained tool that extracts forensic information from a memory dump and presents it in a format that is suitable for further analysis. A comparison of the results obtained with utilities that are commonly employed in live response demonstrates that CMAT provides similar information and identifies malware that is missed by the utilities.

Keywords: Live response, memory analysis, rootkit detection

1. Introduction

When a large enterprise responds to a cyber security incident, it may not be feasible to perform the recommended disconnection and imaging process to gather forensic evidence [13], especially in the case of servers that are critical to business operations. According to CNET [19], Amazon lost $29,000 in revenue per minute during its June 2008 outage. In addition, there have been several recent cases where defendants were found innocent or guilty based on the live response information retrieved at the time of the incident [2]. Clearly, tools that can take quick snapshots of affected computers while minimizing the impact on business operations and preserving evidence are vital to digital forensic investigations. Since many of the items of interest in a live response are maintained in computer memory rather than the hard drive, a mini-

mally intrusive live response would involve the analysis of a live memory capture.

Three steps are involved in developing a minimally intrusive standalone forensic tool for memory: (i) a front-end memory dump routine that directly accesses physical memory; (ii) a memory analysis tool that extracts environmental and activity information from the memory dump; and (iii) a synthesis tool that correlates the environmental information to provide a human-understandable narrative of what occurred on the machine.

This paper focuses on the second step: developing a compiled memory analysis tool (CMAT) that extracts environmental and activity information from a memory dump. CMAT parses a memory dump 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. CMAT is tested against several utilities that extract forensic information from a live system. Experimental results indicate that CMAT provides comparable information as traditional live response utilities, and also uncovers malware that these tools miss.

2. Background

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 the hard drive [24]; retrieving decryption/encryption keys for an encrypted hard drive [24]; and defeating the Trojan horse defense of "I didn't do it, it was the malware installed on my machine" [2]. Valuable live response information that resides in memory includes [4, 5, 10, 24]:

- System date and time
- Logged in users and their authorization credentials
- Network information, connections and status
- Process information, memory and process-to-port mappings
- Clipboard contents
- Command history
- Services and driver information
- Open files, registry keys and hard disk images

One of the arguments against live response is that the evidence capture process modifies the machine state. Several techniques have been proposed to address this issue [1, 3, 9, 22]. These techniques attempt to capture all the memory instead of specific pieces of forensic information. This helps answer the immediate forensic questions and also provides a means to answer additional questions that may come up later in the investigation. The alternatives include hardware-based memory acquisition [1, 3], virtual machines, hibernation files [22] and operating system patches [9].

Hardware-based memory acquisition techniques circumvent the reliance on the operating system by using hardware that interacts directly with memory. These techniques involve the use of custom hardware [3] or the FireWire interface [1] for direct memory access. But the two techniques suffer from the "northbridge exploit" [16]. Since all peripheral devices use the northbridge to interface with the CPU and main memory, malware can be loaded onto the northbridge to subvert memory access by a peripheral while leaving the CPU and operating system untouched. This enables the malware to remain undetected.

The northbridge exploit can be contained by constraining 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. However, one problem with this method is that implementing such a policy within a large organization incurs high costs in terms of time and money. Moreover, software virtualization can slow the perceived responsiveness of a computer because every command has to go through an additional level of abstraction.

Hibernation files offer an alternative memory capture method that relies on the operating system [22]. The process of analyzing hibernation files requires placing the system in the hibernation mode, copying the hibernation files and extracting their contents. However, while the hibernation function is enabled on many laptops, it is disabled by default on most desktops. Desktops can be reconfigured to enable the hibernation mode, but this must be done in advance. Other difficulties include the lack of published descriptions of hibernation file formats and the fact that not everything in memory may be stored.

Libster and Kornblum [9] have proposed an alternative method for acquiring a pristine copy of memory. The method involves 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 sends over a network to a secure machine, and then resume

execution. The difficulty with this method is to ensure that it is deployed for all the disparate systems of a large enterprise prior to an incident.

While a pre-installed operating system patch appears to be an ideal solution in cases where prior access is not possible, the only option available is to initiate a new process to dump system memory. To minimize the impact, the application should limit application program interfaces (APIs) that interface with the operating system and the use of a graphical user interface (GUI). Unfortunately, while many available tools (e.g., [12, 23]) are run from the command line, most use operating system APIs to dump memory. The only exception appears to be Memoryze [11].

Memory analysis tools extract different amounts and types of live response data from a memory dump. The Volatility tool [25] extracts the date and time, running processes, open ports, process-port mappings, strings-process mappings, process-file mappings, process dynamic link libraries (DLLs) and open connections. Additional plug-ins are available to enhance the functionality of Volatility, including the ability to extract registry information. However, Volatility suffers from the limitation that it does not extract port information for all Windows operating system service pack and patch configurations.

Comprehensive open source memory analysis tools written in compiled languages such as C/C++ are not available for Windows XP systems. Betz [21] has developed a tool that finds processes and threads in memory and extracts some information, but it is limited to Windows 2000 systems and does not support the analysis of registry files, network activity and process creators. PTfinder [17] also searches through memory for processes and threads, and handles memory captures from Windows XP systems. However, like Betz's tool, it does not support the analysis of registry files [7], network activity and process creators [8].

3. Compiled Memory Analysis Tool

The compiled memory analysis tool (CMAT) is a C++ command line utility that runs on Linux and Windows operating systems. It parses a Windows XP memory dump to obtain information on user accounts, the Windows registry and running processes. CMAT provides system, process, registry, open ports and user information in a standalone tool that runs without API calls or high-level language interpreters. CMAT operates in two passes. During the first pass, CMAT searches the memory dump file for processes, threads and registry hives. In the second pass, CMAT organizes 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 incorporates data structures needed for the operating system to function. These data structures are useful for extracting live response information. For example, processes are stored in memory as _EPROCESS structures. Potential process structures can be found by searching for strings formatted as a _DISPATCH_HEADER of a _KPROCESS. In the case of Windows XP SP2, this corresponds to a _DISPATCH_HEADER of `0x03` and a _KPROCESS size of `0x1b`. After a likely process is found, it is double-checked to ensure that 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 a process executable or process data, not a process header. This method finds all processes, including normal active processes, hidden active processes (e.g., processes hidden by a rootkit like FUTo [20]), defunct processes and processes from previous boots that have not been removed from memory. In addition, because the _EPROCESS structure also contains a doubly-linked list of all active processes, processes that were missed during the string search can still be found by traversing the linked list. The scan of the linked list also assists in identifying 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 how it searches for processes, CMAT looks for _CMHIVE structures with a signature in Windows XP of `0xbee0bee0` [7]. Just as a process must have a valid memory address for its page directory table base, a registry hive must have a valid memory address for its base block. CMAT also takes advantage of the doubly-linked _CMHIVE structures to ensure that all the registries are found. CMAT then moves through the default user registry to find relationships between session IDs and user names. In Windows XP, it finds the ProfileList key under *\USER\MICROSOFT\WindowsNT\CurrentVersion* and then makes a record of the session IDs and their associated ProfileImagePaths, providing human-understandable names for the users that were logged in.

As a part of its processing, CMAT translates the virtual addresses stored in Windows kernel data structures into physical memory locations. Windows XP has two levels of indirection when it is run on a 32-bit machine; this is achieved by splitting a virtual address into a page directory index, a page table index and an offset. However, if large (4 MB) pages are used, there is one level of indirection; if physical address extensions are used, there are three levels of indirection. While CMAT can detect if large pages are being used, it requires user assistance in the

form of a parameter value to discern if physical address extensions are in use. One method to avoid this is to try alternative virtual-to-physical mappings until the correct mapping is found [25]. Another method is to locate the kernel system variables that store this information [6].

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

4. Test Methodology

The CMAT output was compared with the outputs of five Sysinternals utilities: `psinfo`, `pslist`, `logonsessions`, `handles` and `listdlls` [14]. System information that was examined included the operating system version, number of processors and number of processes. The process information examined included the process creator, files opened, registry keys accessed and modules loaded. The only type of volatile information not examined was network information, including the ports and sockets opened by processes. Also, a comparison was made of the number of distinct DLLs used by ManTech's physical memory dump (`mdd`) utility compared with the number of distinct DLLs used by the Sysinternals utilities.

A Windows XP SP3 system with 2,038 MB RAM was used in the tests. Several application programs were launched on the machine including Internet Explorer, Word, PowerPoint, Visual Studio, Calculator, Kernel Debugger and two command line shells. One of the command line shells was hidden by the FUTo rootkit [20]. The memory dump was collected using ManTech's `mdd` utility (version 1.3), which required 67 seconds.

5. Test Results

The test results in Table 1 demonstrate that CMAT provides the same or equivalent information as the Sysinternals utilities. In addition, CMAT found the process hidden by FUTo while the Sysinternals utilities did not. The Sysinternals utilities failed to associate several processes with the users that started them. Except for the inability of CMAT to provide network information, CMAT's performance in the tests was exemplary.

- **System Information:** CMAT and `psinfo` provide the operating system version, major and minor version, service pack number

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) vs. 48 (Sysinternals)	

and build number. Also, they both provide the number of processors. The `psinfo` utility also provides the processor type and speed along with the video driver used. However, this information appears to have limited forensic use; therefore, this feature was not implemented in CMAT. The one additional piece of information provided by `psinfo` compared with CMAT is if a physical address extension is used (as described above, this information is passed to CMAT as a parameter). A future version of CMAT will address this shortcoming.

- **Processes:** CMAT reported 58 active processes while `pslist` reported 57 processes; 56 of the processes matched (97% equality). The program missing from the CMAT results was `pslist`, which was not running when the memory was dumped. Similarly, the program `mdd_1.3` was missing from the `pslist` result, which makes sense because the memory was already dumped when `pslist` was executed. The other program missing from the `pslist` result was `cmd.exe`, which was hidden by the FUTo rootkit; this demonstrates that a program intentionally hidden 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 separate utility (`logonsessions`) for this purpose. Comparison of the CMAT and `logonsessions` results showed that nine processes were not listed by `logonsessions`, including all the processes owned by LocalService and NetworkService, and four processes owned by SystemProfile (including the system and idle processes). Identifying all the system processes is vital to detecting if malicious services were executing on the machine.

- **Loaded Modules:** CMAT and the Sysinternals `listdll` utility provided identical lists of loaded modules (100% equality).

- **Files and Registry Keys:** The lists of files and registry keys provided by CMAT and the Sysinternals `handle.exe` utility matched, except for the temporary files that were opened and closed between the memory dump and the execution of `handle.exe`. These files were listed as having `[RWD]` access. CMAT identified file handles that were actually device handles (e.g., *\Device\KsecDD*), but was unable to print the names of the devices. Similarly, while `handle.exe` summarized the long code for the current user with a succinct HKU, CMAT displayed the entire registry key.

- **Dynamic Link Libraries:** As mentioned above, minimizing the number of loaded modules (DLLs) used when performing a live response is desirable because less evidence on the hard drive is modified. This test compared the numbers of DLLs used by ManTech's `mdd` memory capture tool and the five Sysinternals utilities. To collect the data, `mdd` was executed during the time that each of the five Sysinternals utilities was running. The DLL lists associated with the five Sysinternals utilities were extracted from the memory dumps using CMAT; duplicate DLLs among the five utilities were removed and the results were then tallied. The results show that performing a memory capture instead of a live response significantly reduces the impact on the evidence. In particular, the memory capture used 15 DLLs while the five Sysinternals utilities in total used 48 different DLLs, corresponding to a 69% reduction in system impact.

6. Conclusions

The CMAT live response tool provides comparable information from a memory dump and produces a 69% less impact on system state (potential evidence) than the popular Sysinternals utilities. However, CMAT is unable to extract several pieces of useful forensic information, including the mapping of ports and sockets to processes (provided by the Sysinternals `portmon` utility). The difficulty arises because this information is stored in the data section of the TCP/IP module. While the port and socket data reside in specific locations for different Windows operating system versions, service packs and patch configurations, their locations can change when new patches are installed.

A future version of CMAT will target Microsoft Windows XP operating systems. One difficulty is to correctly map the kernel data structures for different Windows versions (e.g., Vista and Windows 7). This difficulty can be addressed by retrieving the operating system data structures dynamically after the operating system is identified upon parsing

the memory dump. Microsoft has recognized the need for such flexibility by providing the data structures (symbol tables) for download in real time to support its kernel debugger. CMAT will attempt to make use of these symbol tables to facilitate memory analysis for different versions of the Windows operating system.

References

[1] A. Boileau, Hit by a bus: Physical access attacks with FireWire (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, Center for Education and Research in Information Assurance and Security, Purdue University, West Lafayette, Indiana, 2005.

[3] B. Carrier, *File System Forensic Analysis*, Pearson, Upper Saddle River, New Jersey, 2005.

[4] B. Carrier and J. Grand, A hardware-based memory acquisition procedure for digital investigations, *Digital Investigation*, vol. 1(1), pp. 50–60, 2004.

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

[6] B. Dolan-Gavitt, Finding kernel global variables in Windows (moyix.blogspot.com/2008/04/finding-kernel-global-variables-in.html), April 16, 2008.

[7] B. Dolan-Gavitt, Forensic analysis of the Windows registry in memory, *Digital Investigation*, vol. 5(S), pp. S26–S32, 2008.

[8] B. Dolan-Gavitt, Linking processes to users (moyix.blogspot.com/2008/08/linking-processes-to-users.html), August 16, 2008.

[9] E. Libster and J. Kornblum, A proposal for an integrated memory acquisition mechanism, *ACM SIGOPS Operating Systems Review*, vol. 42(3), pp. 14–20, 2008.

[10] K. Mandia, C. Prosise and M. Pepe, *Incident Response and Computer Forensics*, McGraw-Hill/Osborne, Emeryville, California, 2003.

[11] Mandiant, Memoryze, Washington, DC (www.mandiant.com/software/memoryze.htm).

[12] ManTech, Memory DD, Vienna, Virginia (cybersolutions.mantech.com/products.htm).

[13] National Institite of Justice, Electronic Crime Scene Investigation: An On-the-Scene Reference for First Responders, U.S. Department of Justice, Washington, DC, 2009.

[14] M. Russinovich, Sysinternals Suite, Microsoft Corporation, Redmond, Washington (technet.microsoft.com/en-us/sysinternals/bb842062.aspx).

[15] M. Russinovich and D. Solomon, *Microsoft Windows Internals*, Microsoft Press, Redmond, Washington, 2005.

[16] J. Rutkowska, Beyond the CPU: Defeating hardware-based RAM acquisition (Part I: AMD case), presented at the *Black Hat DC 2007 Conference* (www.first.org/conference/2007/papers/rutkowska-joanna-slides.pdf), 2007.

[17] A. Schuster, PTfinder (version 0.2.00), Bonn, Germany (computer.forensikblog.de/en/2006/03/ptfinder_0_2_00.html), 2006.

[18] A. Schuster, Searching for processes and threads in Microsoft Windows memory dumps, *Digital Investigation*, vol. 3(S), pp. S10–S16, 2006.

[19] S. Shankland, Amazon suffers U.S. outage on Friday, CNET, San Francisco, California (news.cnet.com/8301-10784_3-9962010-7.html), June 6, 2008.

[20] P. Silberman, `FUTo`, *Uninformed*, vol. 3 (www.uninformed.org/?v=3&a=7&t=sumry), January 2006.

[21] SourceForge.net, Memparser (sourceforge.net/projects/memparser), 2006.

[22] M. Suiche, Sandman Project (sandman.msuiche.net/docs/SandMan_Project.pdf), 2008.

[23] M. Suiche, `win32dd` (win32dd.msuiche.net).

[24] I. Sutherland, J. Evans, T. Tryfonas and A. Blyth, Acquiring volatile operating system data: Tools and techniques, *ACM SIGOPS Operating Systems Review*, vol. 42(3), pp. 65–73, 2008.

[25] A. Walters and N. Petroni, Volatools: Integrating volatile memory forensics into the digital investigation process, presented at *Blackhat Hat DC 2007 Conference* (www.blackhat.com/presentations/bh-dc-07/Walters/Paper/bh-dc-07-Walters-WP.pdf), 2007.