Providing a Foundation for Analysis of Volatile Data Stores

Timothy Vidas
Naval Postgraduate School
Monterey, CA
tvidas@nps.edu

ABSTRACT

Current threats against typical computer systems demonstrate a need for forensic analysis of memory-resident data in addition to the conventional static analysis common today. Certain attacks and types of malware exist solely in memory and leave little or no evidentiary information on nonvolatile stores such as a hard disk drive. The desire to preserve system state at the time of response may even warrant memory acquisition independent of perceived threats and the ability to analyze the acquired duplicate.

Tools capable of duplicating various types of volatile data stores are becoming widely available. Once the data store has been duplicated, current forensic procedures have no method for extrapolating further useful information from the duplicate. This paper is focused on providing the groundwork for performing forensic investigations on the data that is typically stored in a volatile data store, such as system RAM.

It is intended that, when combined with good acquisition techniques, it will be shown that it is possible to obtain more post incident response information along with less impact to potential evidence when compared to typical incident response procedures.

Keywords: Digital Forensics, Volatility, RAM, Windows Forensics, Computer Memory, Acquisition

1. INTRODUCTION

A common incident response step taken early in the process is to 'pull the plug' from a powered on machine (Secret Service, 2002). Practitioners recognized that performing a 'clean' shutdown could further change the state of the system. However 'pulling the plug' also has its own drawbacks on later analysis. One such drawback is the lack of ability to identify and examine the execution state of the machine at the time of seizure.

Some tools allow the acquisition of the contents of 'raw' RAM from a running system (Shipley & Reeve, 2006). Thus far, the analysis of a RAM image using commercial tools has been limited to small special-use devices such as PDA's or various cellular phones. For most forensic cases seen today, traditional

post-mortem techniques may be sufficient for the United States court process, but for cases involving an active adversary or completely memory resident threat (such as some viruses and worms), analysis of volatile data stores will not only be recommended, but will be required.

Regardless of the effectiveness and completeness of the methods and mechanisms used for the acquisition of volatile data stores, procedures need to be created to perform media analysis akin to those in use today for traditional media.

This paper focuses on the analysis of the different portions of RAM used by mainstream operating systems in order to adapt current response methodologies to promote further preservation of the state of a suspect system. It is intended that, given a complete RAM capture, the amount of information available after the initial response be equal to or greater than the information that could have been obtained using current incident response procedures. Furthermore, the acquisition of RAM would have imposed less negative impact to the preservation of evidence.

2. BACKGROUND

Currently, after an incident, captured memory (if available) is analyzed using techniques that would be considered crude if used for traditional file system level forensics. A simple hex view or strings (Fedora Core 4, 2006) analysis may be used by an investigator to simply glance at a subset of data looking for something that might provide some direction (Stover & Dickerson, 2005).

Experiments run in conjunction with this project showed that, on average, a cleanly install and booted contemporary Windows based workstation with 512 MB of RAM would produce largely unusable strings output. Unusable does not suggest that a string such as "dollar" was found, but it was not pertinent because this was not a counterfeiting suspect. Unusable indicates that, while technically printable, most of the strings extracted have no inherent meaning, such as "EWCcedh". The ratio of the amount of information obtained from the data is very low. The situation is only worsened if only a hex view analysis is performed without the aid of a tool such as strings.

Recently the research community has made some forward progress in RAM forensics. The 2005 Digital Forensic Research Workshop (DFRWS) challenge served as a launch pad for academic research (DFRWS, 2007). Unfortunately, the challenge produced no open tools and little insight into the methodologies. A few topically related projects have become openly available, among them are Procloc (Vidas, 2007), IR/CF Tools (Carvey, 2007), WMFT (Burdach, 2007), and Ptfinder (Schuster, 2007).

3. COMPARISON TO TRADITIONAL METHODS

The analysis of volatile stores and traditional postmortem forensics vary

greatly. Regardless of the medium, traditional forensics typically involves the postmortem media analysis of a file system.

3.1 Lack of Structure

Though it is common to speak of analyzing a particular workstation or personal computer, the analysis is very often focused on a file system. Even 'advanced techniques' focus on clarifying or adding to the file system that is being examined.

Popular industry products can perform automated actions such as recovering folders, finding partitions, undeleting items, etc. All of these actions work toward the goal of having an "evidence container" in which to perform analysis. All further analysis is done within this container.

Consider a word processor document that contains an embedded digital picture. Contemporary tools allow an analyst to quickly view all digital pictures on the media, including the embedded picture. This picture does not in itself exist as a file, but as a portion of a file; however the picture by itself may be considered as evidence. Even the "bit-for-bit" duplicates of hard disks are parsed and data that was not contained within the suspect file system on the original disk is added to the evidence container in the analysis software (such as certain 'deleted' files). These types of files are added to the container as additional 'items.' Thus traditional analysis depends very heavily on the understanding of the file system that was used on the suspect system, and the file system is the primary focus of analysis.

Volatile data stores typically have no file system abstraction layer and the data within is managed directly by the operating system. For this reason, tools that focus on the analysis of file systems are not able to cope with volatile stores well. When considering a volatile data store such as Random Access Memory (RAM), factors other than the file system become main focuses: the operating system type and version in use, the configuration of that operating system and possibly other information such as specific hardware in use.

3.2 Acquisition

When acquiring a hard disk drive, it is preferred that the drive is not in use at the time of acquisition. Acquisition procedures may even dictate disconnecting the drive and using special hardware such as a write-blocking device. These procedures are in place to ensure that the input to the duplication function is unvarying and thus establish reproducible duplication results. When considering memory, current acquisition techniques actually involve the use of the host system. Creating the duplicate changes the contents of memory.

Particular instances of volatile stores will typically vary much more than instances of non-volatile stores. This variance is partly due to the changing nature of volatile stores like RAM, which is perceived as a faster, more

valuable resource than a non-volatile store to the system and is thus always in contention. This resource contention results in a further inability to acquire a complete point in time duplicate of RAM as is possible with nonvolatile stores such as a Hard Disk Drive. While work is being done on developing special hardware to facilitate a more pristine copy, this work is not yet readily available and when such hardware does exist it will still require pre-incident installation (Carrier & Grand, 2004).

Contrary to popular belief, data may still exist in a volatile data store from a time prior to the last reboot of the system (Chow et all, 2005) (which actually challenges the term 'volatile'). While most hardware is capable of "zero-ing" or otherwise clearing the contents of RAM at boot, many systems ship with the default setting to "quick" mode where no memory testing or clearing is performed at all. It should be pointed out that this capability is usually presented at a level much lower than the operating system, typically as a BIOS feature.

4. ANALYSIS

In order to demonstrate the value of capturing volatile data, it must be shown that given a duplicate of a volatile store, that at least as much information must be attainable as would have been attainable via typical incident response procedures, with as little impact to the state of the volatile store as possible.

It is prudent to point out that all current methods of volatile data acquisition actually alter the state of the volatile data. For example when duplicating RAM, a new process must be created to perform the act of duplication. The creation of this process will alter the state of the RAM.

While it is ultimately desirable to capture and analyze all portions/types of volatile data (i.e. processes, threads, current network connections, open files, etc), for demonstrative purposes, it must be shown that it is viable to perform post incident analysis of at least one portion. The analysis must produces at least as much information as a comparable live response tool. Simulation of the Windows Task Manager was selected to demonstrate proof of concept. Or for those familiar, PSList.exe as part of the Sysinterals toolkit provides a closer approximation.

As with most Digital Forensics cases, the procured information will be interpreted with respect to the unique case (i.e. counterfeiting vs malware creation vs identity theft, etc). It may very well be the case that analysis of all the above mentioned portions/types of memory are not worthy of noting in all cases. If the volatile store was acquired in its entirety, running multiple tests or performing repeated analysis will not further taint evidence.

4.1 Implementation Goals

Several criteria easily stand out as being desirable when implementing such a tool. In an attempt to gain acceptance the main focus of the initial tool was to not only make it functional, but to encourage use by end-users. Primary criteria included:

1. Must work on dd-style dumps (preferred) and on Microsoft DMP 'Complete' style dumps.
2. Must be simple to use.
3. Must accurately produce results that would have normally been obtained by running commands during incident response. (for tool development it must accurately re-produce a pre-response observed set of processes)
4. Must work on multiple versions of Windows (and be adaptable to Linux).

4.2 Brute Force Searching

When parsing a memory image, the most complete way to search for process structures will be to start assuming each byte is the first byte of a process structure and validating (or not) the assumption with tests, then to shift one byte and repeat the process. Such a search would be considered a linear brute force search.

Even though each EPROCESS structure contains pointers to other EPROCESS structures (Windows maintains a doubly linked list), it is preferable to manually locate EPROCESS structures so that the results can include both latent processes and potentially, processes attempting concealment from tools that enumerate processes (such as the Task Manager utility).

When considering the increasingly large amounts of RAM available in today's OSs, a linear byte-by-byte search may very well be considered computationally impractical. A search assuming the initial byte must be page aligned would be much faster. While it is not generically safe to assume that all process structures will be allocated on a page boundary, it may be safe to assume certain other boundaries greater than one (such as an eight byte boundary for Windows)(MSDN, 2007). If a particular OS implements certain boundaries, it may be possible to search based on these offsets in order to greatly reduce the amount of testing and thus processing.

4.3 Structures of Interest

Processes and threads are vital concepts required to be explored in light of the objectives. Even though internal structures are by definition not known in closed source products such as Windows, methods such as debugging and

reverse engineering can be used to gain insight about these structures. Some of these structures are explored in the following few sections.

4.3.1 EProcess Structure

The Windows process structure, EPROCESS, can be enumerated using a kernel debugger. (e.g. using the Windows debugger to enumerate fields by issuing a `!proccessfields`, `dt _eprocess` or dt `nt!_eprocess` command.) (Microsoft Corp., 2006) Substructures can also be enumerated in this way. From the information gleamed from the debugger a signature for the EPROCESS and subsequent structures can be created. Other substructures such as the Kernel Process KPROCESS (Process Control Block - PCB) can be modeled similarly, and some EPROCESS elements, such as the Process Environment Block (PEB), are pointers to data that exists elsewhere. A listing and description of EPROCESS members can be found in Microsoft Windows Internals (Russinovich & Solomon, 2006).

Certain parts of the EPROCESS structure stand out as being easily identifiable. Similar to how file remnants and certain deleted files are found in unused portions of a file system or disk device, it is possible to find EPROCESS structures by locating individual portions of the structure and then testing other sections (by offset, since offsets can be discerned from the structure dump) of the EPROCESS candidate for validity.

4.3.2 EProcess substructure Timestamp

One part of the EPROCESS structure that may be easy for the reader to relate with is the timestamp information. While most readers will be familiar with the concept of a timestamp, many may not be familiar with this particular implementation. FILETIME is a Windows defined structure that has existed since Windows 3.1 but is also defined in the current .NET framework 2.0. It is a 64 bit value that consists of two data members: the high order 32 bits are `dwHighDateTime` and the low order 32 bits are `dwLowDateTime`. The 64 bits typically represent a number of 100 nanosecond intervals since January 1, 1601. Once the FILETIME portion of an EPROCESS is known, some conversion must take place to make this a usable timestamp for investigative purposes. A benefit of decoding a timestamp allows for comparison of disk times to process times for rough estimating and correlation (MSDN, 2006).

Below are two offsets from a Windows XP SP2 EPROCESS structure. It is easy to see the Low and High order sections and in fact the 64 bit math required to decode this into a human readable timestamp is fairly straightforward. Depending on the debugger options, the offsets will be reported as:

```
+0x070 CreateTime    : _LARGE_INTEGER
  +0x078 ExitTime      : _LARGE_INTEGER
```

or in a more detail with the same tool as:

```
    +0x070 CreateTime   : union
      _LARGE_INTEGER, 4 elements, 0x8 bytes
      +0x000 LowPart    : Uint4B
      +0x004 HighPart   : Int4B
      +0x000 u   : struct
        __unnamed, 2 elements, 0x8 bytes
        +0x000 LowPart  : Uint4B
        +0x004 HighPart : Int4B
      +0x000 QuadPart   : Int8B
    +0x078 ExitTime     : union
       _LARGE_INTEGER, 4 elements, 0x8 bytes
      +0x000 LowPart    : Uint4B
      +0x004 HighPart   : Int4B
      +0x000 u          : struct
         __unnamed, 2 elements, 0x8 bytes
        +0x000 LowPart  : Uint4B
        +0x004 HighPart : Int4B
      +0x000 QuadPart   : Int8B
```

4.3.3 EProcess substructure Process Control Block

The very first portion of the EPROCESS structure is the PCB and at the first offset a header can be found.

```
+0x000 Pcb  : struct _KPROCESS, 29 elements, 0x6c bytes
     +0x000 Header  : struct
            _DISPATCHER_HEADER, 6 elements, 0x10 bytes
        +0x000 Type             : UChar
        +0x001 Absolute         : UChar
        +0x002 Size             : UChar
        +0x003 Inserted         : UChar
        +0x004 SignalState      : Int4B
        +0x008 WaitListHead     : struct
            _LIST_ENTRY, 2 elements, 0x8 bytes
           +0x000 Flink            : Ptr32 to
           +0x004 Blink            : Ptr32 to
        +0x010 ProfileListHead  : struct
            _LIST_ENTRY, 2 elements, 0x8 bytes
           +0x000 Flink            : Ptr32 to
           +0x004 Blink            : Ptr32 to
```

The 16 header bytes specify the type of structure that follows. (The same header is used not only by processes and threads but also events, semaphores, queues, etc.). (Russinovich & Solomon, 2005)

Some processes may seem to share the same or very similar values for locations (such as the Process Environment Block). These addresses are typically virtual addresses and the distinction between processes can be shown by converting the virtual address to the physical address.

4.3.4 The Dispatch Header

For each candidate structure a Dispatch Header is assumed, then data members of the Header can be checked, offsets to other sections of the EPROCESS or ETHREAD can be checked, and values of certain fields can be checked because ranges of values for these fields are known (such as date, process priority range, existence of a Page Directory Index, or kernel memory address which must be mapped above 0x80000000).

`EPROCESS` structures can be found in different versions of Windows by utilizing different offsets for equivalent portions of the EPROCESS structure. For example, a Windows XP SP2 EPROCESS structure contains the Process ID (PID) at offset `0x09c`, while Windows 2000 SP4 EPROCESS structure contains the PID at offset `0x084`.

Windows XP SP2 EPROCESS field:

```
+0x09c UniqueProcessId  : Ptr32 to
```

Windows 2000 SP4 EPROCESS field:

```
+0x084 UniqueProcessId  : Ptr32 Void
```

Different types of structures, such as a thread, can be located using techniques similar to those used for locating processes. Known offsets can be used to validate fields of potential structures.

4.4 Duplicate Type Compatibility

Several methods could be used to provide compatibility between the complete and dd style memory dumps. Since the complete style memory dump contains a header in addition to the RAW memory data, the complete memory dump could be 'converted' to a dd style dump by removing the header. Similarly, during processing the header could simply be ignored by skipping to the offset pertaining to the first memory location. This skip will only introduce minimal overhead (such as having to subtract out the amount of the skip when reporting structure location in RAM).

Finally, a complete memory dump may actually be processed identically to that of a dd style dump as long as the DMP header size is a multiple of the page size (or in the case of this particular research, the first location of to the contents of the RAM dump falls on the aforementioned 8 byte scan boundary.)

It is prudent to point out that the DMP file format is proprietary and the above discussion is based on both observation and assumption.

4.5 Non-Volatile Store Correlation

Much information either required for or beneficial to the analysis of volatile stores will likely only be attainable from the non-volatile stores. Size and format of data structures, method of segmentation and management, even page size may depend upon OS version and/or hardware in use. Even the owner of a process is tracked in a process structure as a portion of an access token, which would have to be compared with registry entries in order to obtain the associated username. Therefore obtaining information from a non-volatile store from the suspect system version (i.e. the version of OS from the hard disk) can be quite beneficial for the analysis of the volatile store.

Even if it is not a technical requirement, having some information typically acquired using traditional non-volatile techniques, or in some cases live response steps, may serve as an enabler for analysis on acquired volatile data. OS type and patch level are among the foremost important factors.

Other types of information from non-volatile stores may prove to be very valuable. Correlation a RAM duplicate with the pagefile on disk would yield a more holistic view of virtual memory.

5. CONCLUSIONS

Fully commented implementation program written in PERL employing five tests for both process and thread structures for Windows 2000 through Windows 2003 Server is less than 1000 lines of code. The script can fully parse a 512 MB RAM image in about 7 minutes when executing on a Pentium 3m with 1 GB of RAM.

Potentially every member of the EPROCESS structure could be checked for validity using one or more tests per member, and each test could be ranked in order to create a heuristic for determining accuracy. History typically shows that more tests should produce more accurate results. However experiments on controlled, baseline RAM duplicates demonstrate high accuracy with as few as five implemented tests.

The implementation confirms that information about the state of a system can be found postmortem. At the very least, Task Manager functionality can be simulated by locating EPROCESS structures in a RAM image, and in some cases more information is available than Task Manager is capable of reporting (such as an "old" process). Furthermore, good acquisition techniques can provide this information with less impact than using similar tools on a live system. Unfortunately, this does not give a responder the ability to alter the response based on the state in which the system is found, but does allow the

state of the system to be preserved along with the preservation of the non-volatile stores, and inspected at a later time. Further preserving system state and the capability to re-inspect RAM contents at a later time are both desirable abilities, and should warrant the consideration of RAM acquisition as part of incidence response.

6. FUTURE WORK

The current state of the tool should definitely be considered beta and should not be construed as production level. It could easily be improved or redesigned to be more likely to be adopted by mainstream responders. Desirable features may include: automatic detection of the OS from which the RAM was acquired, detection of popular dump formats (DMP) versus dd-style RAM capture, automated extraction of selected/certain processes memory space, pagefile unification, automated store correlation (i.e. registry hives), supporting boot switches such as `/3G` and `/PAE`, and supporting architectures other that i386.

Similar to how hash lists are used today for known good or known bad files, lists or heuristics could be added in order to bring attention to objects that are likely to require further research. In order to find outliers, the set of processes found via brute-force could be compared with those found in the linked list that is maintained by the OS. Processes and threads that "don't play by the rules" could be flagged as well (i.e. no window title, path, etc)

The current version of the analysis tool is limited to process related information. The creation of similar tools to obtain other popular incident response information (like current network information, open files, etc) should be explored.

7. ACKNOWLEDGEMENTS

Some related preliminary work was previously presented at the Third Annual IFIP WG 11.9 International Conference on Digital Forensics in Orlando, FL on January 28-31, 2007.

8. REFERENCES

Burdach, M. (2007), Forensic Analysis, http://strony.aster.pl/forensics/, Accessed Jan 10, 2007.

Carrier, B., Grand, J. (2004) “A Hardware-based Memory Acquisition Procedure for Digital Investigations,” Digital Investigation. Vol1 (Issue 1):50-60

Carvey, H. (2007), Windows IR/CF Tools, http://sourceforge.net/projects/windowsir, Accessed Jan 10, 2007.

Chow J., Pfaff B., Garfinkel T.,and Rosenblum M. (2005) 'Shredding Your Garbage: Reducing Data Lifetime Through Secure Deallocation'. 14th USENIX Security Symposium. July/August 2005. Baltimore, MD.

DFRWS (2007), DFRWS 2005 Forensics Challenge, http://dfrws.org/2005/challenge/, Accessed Jan 10, 2007.

Fedora Core 4 (2006). 'Strings man page,' Fedora Core 4.

KB 555223 (2007),'RAM, Virtual Memory, Pagefile and all that stuff,' http://support.microsoft.com/default.aspx?scid=kb;en-us;555223, Accessed Jan 10, 2007.

Microsoft Corp. (2006), Debugging Tools for Windows help file. Microsoft Corp.

MSDN (2007), '.NET Framework FILETIME specification,' http://msdn2.microsoft.com/en-s/library/system.runtime.interopservices.comtypes.filetime.aspx, Accessed Jan 10, 2007.

MSDN (2007), 'Six tips for efficient memory usage,' http://www.microsoft.com/whdc/driver/perform/mem-alloc.mspx, Accessed Jan 10, 2007.

MSDN (2007), 'Why you cant tread a FILETIME as an int64,' blogs.msdn.com/oldnewthing/archive/2004/08/25/220195.aspx, Accessed Jan 10, 2007.

Russinovich, M. and Solomon, D (2005). Microsoft Windows Internals. Fourth Edition. Microsoft Press. Redmond, Washington.

Schuster, A. (2007), PTFinder Version 0.3.00, http://computer.forensikblog.de/en/2006/09/ptfinder_0_3_00.html, Accessed Jan 10, 2007.

Shipley, T. and Reeve, H. (2006), Collecting Evidence from a Running Computer: A Technical and Legal Primer for the Justice Community. The National Consortium for Justice Information and Statistics.

Stover, S. and Dickerson, M. (2005), 'Using Memory Dumps in Digital Forensics,' ;Login: The USENIX Magazine. Volume 30, Issue 6.

United States Secret Service (2002), Best Practices for Seizing Electronic Evidence. Second Edition.

Vidas, T. (2007), NUCIA, http://nucia.unomaha.edu/tvidas/, Accessed Jan 10, 2007.