Currently "In Submission" to JDFP
(some content may change before publication)

THE ACQUISITION AND ANALYSIS OF RANDOM ACCESS MEMORY

Timothy Vidas
Naval Postgraduate School
Monterey, CA

ABSTRACT
Mainstream operating systems (and the hardware they run on) fail to purge the contents of portions of volatile memory when that portion is no longer required for operation. Similar to how many file systems simply mark a file as deleted instead of actually purging the space that the file occupies on disk, Random Access Memory (RAM) is commonly littered with old information in unallocated space waiting to be reused. Additionally, RAM contains constructs and caching regions that include a wealth of state related information. The availability of this information along with techniques to recover it, provide new methods for investigation.

This article discusses the benefits and drawbacks of traditional incident response methods compared to an augmented model that includes the capture and subsequent analysis of a suspect system's memory, provides a foundation for analyzing captured memory, and provides suggestions for related work in an effort to encourage forward progress in this relatively new area of digital forensics.

KEYWORDS: memory, random access memory, memory analysis, digital forensics, Windows forensics, incident response, best practices

Tim Vidas is a Research Associate at the Naval Postgraduate School. He has been focusing research in the field of digital forensics for a few years and is now primarily working on in the area of trusted operating systems and kernels. In addition to research, he likes to teach and has a wide set of IT related interests. He maintains several affiliations like ACM, CERT, and Infragard and holds several certifications such as CISSP, Sec+ and EnCE. Tim has a BS and MS in Computer Science. In his free time he toys around with forensic competitions and CTF exercises.

A short version of this work was presented at the Third Annual ifip WG 11.9 International Conference on Digital Forensics in Orlando, Florida on Jan 28-31, 2007.

INTRODUCTION

Techniques described here tend to follow a more historical thought process regarding forensic procedures: acquire first, then identify. This may cause some privacy concerns when contrasted with some more modern approaches to e-discovery[1] where the pertinent information is located first and then only that information is acquired. This distinction is also pertinent when considering the classification of information. Traditionally acquired data will need to be classified at the highest classification level of any information found on the system. Theoretically, when using more selective methods of e-discovery, the acquisition could be limited to only acquire data of a certain classification level and thus not be subjected to the high watermark. Both the historical and selective techniques have their benefits and drawbacks; such as completeness versus speed and storage advantages respectively. This text does not debate these techniques.

This article makes many hardware and software assumptions. Intel i386 / IA-32 architecture is assumed, along with a standard 4K page size. Only Microsoft Windows® operating systems are discussed, and for systems that support memory related boot switches such as /3G and /PAE it is assumed that these switches are not being used[2].

Even though the concept of object-reuse and related techniques for its mitigation have been known for decades [1], many operating systems use memory management techniques that have little or no safeguards against this threat. A design decision to place a higher precedence on performance than security is not uncommon. In the case of Random Access Memory (RAM), this design choice can be exploited in order to both further preserve and gain deeper insight into the state of a currently running machine.

BACKGROUND

Depending on the situation, upon arriving *on scene*, a responder has two core choices: either interact with the system or pull the plug. On one side, it has been known for some time that normal user interaction is undesirable, even performing a *clean* shutdown would destroy potential evidence by changing timestamps and potentially overwriting information. Following this train of thought, it was suggested that pulling the plug of a machine will leave it in a more preserved state than powering it down gracefully [2] (albeit some subsystems, such as the file system, may not recover gracefully from abrupt removal of power). On the other side, while pulling the plug does preserve the current contents of the hard disk drive, it allows little or no insight into what operations the system was performing at the time when the power was removed. In light of this lack of knowledge, others have provided incident response steps to perform in order to gain insight about the state of the system [3 among others].

When concerned with the contents of RAM, neither choice is adequate. Simply, pulling the plug can clear the contents of RAM (in most cases), and performing many incident response actions overwrites potential evidence in memory akin to creating new files on a suspect hard disk drive. Two additional concepts need to be introduced into acquisition and analysis stages in order to take advantage of RAM contents: the acquisition of RAM, and the extraction of information from the RAM duplicate.

For some time now, varying abilities of acquiring RAM contents have been available. A popular open source LiveCD called Helix has supported George Gartner's dd tool in combination with the windows \\.\PhysicalMemory object since about 2004.

In many cases, this packaging of a tool with the memory object in a mostly graphical form can enable mainstream first responders to capture memory.

Regardless of the method used to acquire memory, little effort has been devoted to the problem of what to do with the copy once is has been acquired. The lack of analysis capability is likely why RAM content is not captured as a matter of course. Prior to 2005, the primary method of analyzing a RAM copy was to perform a strings analysis. In 2005, the Digital Forensics Research Workshop (DFRWS) held a Memory Analysis Challenge which will almost certainly be considered the beginning of the field of memory forensics. Two individuals were credited with winning the challenge (Garner and Betz) but neither publicly released their tools. Since, others have created tools publicly (Vidas, Carvey, Burdach, Schuster [4-7]) and privately (Kornblum, Goldsmith). Current tools have distinct drawbacks, but the future outlook looks promising.

THE CASE FOR COPYING RAM

For those that currently do not copy RAM as part of their acquisition procedures, a logical first question to ask is "Why copy RAM?" There are several reasons that a complete RAM capture may prove useful, most revolve around key differences between data stored in RAM and data stored on a hard disk drive.

Volatile memory, e.g. RAM, is perceived to be more trusted than non-volatile memory, e.g. ROM, magnetic and optical storage. When simply considering the data that is either not stored or somehow protected on a hard disk drive yet stored in plaintext when stored in memory, many data types immediately come to mind: passwords, financial transaction information, encryption keys, etc. The existence of this type of information may not even be intentional, poorly written applications can leave information resident, or this type of information may even occur as a byproduct of malware. Circa 1994, malware sophistication had grown to the point where a multipartite, stealthy virus [8] could slowly encrypt a hard disk drive unbeknownst to the user.

Malware can be completely memory resident. Rather than debate the differences between viruses, worms, trojans, etc. It is sufficient to say that malware can exist completely in RAM. In such a situation the malware may not ever even touch the hard disk drive. After removal of power from the system, no record of the malware would exist upon later examination. Contemporary examples of this would include the widely publicized nimda and SQLslammer worms [9,10].

Memory is latent. Much as a latent fingerprint is one that existed but was not readily evident, there is latent information available in memory. Similar to how the recovery of deleted files became a widespread act early in the field of digital forensics, the recovery of prior (deleted) processes has become a focus of current research in memory forensics. Due to file system caching, delayed writes, buffers, etc. it is even possible to extract full or fragments of files from memory, data that may have never been written to the hard disk drive.

The *hacker defense* is becoming more common [11-17 among others]. Envision a suspect that has known contraband stored on their hard disk drive. A defense mechanism may be to download some malware purposefully. This malware need not even be related to the contraband data in any way. A judge/jury may be convinced that due to the presence of malware and the inability to discern whether the malware could be at fault that a guilty suspect be deemed innocent. The capture of memory can give the ability to both

determine if that the malware in question was actually executing and if so, it may be possible to distinguish the capabilities of the malware in order to meet this burden of proof.

Executing code must actually exist somewhere. Malware routinely relies on obfuscation and other techniques to avoid detection and eradication. However, all code executing on a processor has to actually exist in executable form somewhere. In some cases memory acquisition may prove to be a useful way to perform malware analysis. One example may be executable packing. When executables are packed (binary obfuscation) they are inherently harder to understand. In some situations unpacked versions of executables could be extracted directly from memory in order to avoid tedious and time consuming manual unpacking.

Duplicating RAM has less impact to potential evidence than normal incident response. During incident response, in order to gain insight about system state one might issue several commands and catalog the responses. Typical response may include creating more than 30 processes [3]. The more detailed the responses the more accurate the portrayal of the system state, but the portrayal depends upon the granularity of the tools and the accurate recording and interpretation of the tool output. When considering a copy of RAM as an alternative, the recording is complete, and the interpretation and granularity can be altered via subsequent examination of the copy, a leisure that is not possible via live response.

Why wouldn't you acquire RAM? Even though under most circumstances the actual act of copying RAM will be shown to have a negative impact to potential evidence, the impact should be outweighed by potential gain. Good procedures and documentation should help minimized the effect of potential damage to evidence, and eventually RAM acquisition will become an industry best practice.

It will be shown that similar to Windows Task Manager listing current processes, forensic tools can be (and have been) created that list not only processes active at the time of memory acquisition, but also show old and hidden processes.

RAM ACQUISITION

When creating a duplicate of a hard disk drive, ideally the drive is disconnected from the system and duplicated via a hardware write blocker. Even though power is removed, the data stored on the drive is not lost because the store is non-volatile. This is not the case with volatile memory such as RAM. Due to physical architecture, once power has been removed for a certain amount of time the state of the data in RAM in unknown. This prohibits the removal of RAM chips for duplication, and encourages live acquisition (while the system is running).

The actual acquisition of RAM can be performed in different ways, each with benefits and drawbacks. The biggest difference in technique is hardware vs software acquisition. Currently there are three software based techniques and two hardware based techniques.

Software Acquisition

Software techniques are currently the most prevalent. A tool (such as dd) can be used from a LiveCD (such as Helix) to copy RAM[3]:

```
dd if=\\.\Device\PhysicalMemory
   of=e:\memoryimage.dd bs=4096
```

In this case of software acquisition, some memory (potential evidence) will be over-written because the copy utility itself will be instantiated as a process on the suspect system and the data that was in the portion of memory

that this new process occupies will be lost[4]. For this reason, the footprint of any acquisition tool should be minimal. In the above example Helix was mentioned due to its prevalence in the field, however, the default configuration of Helix may not be conducive to acquisition needs. Helix will start an autorun process called helix.exe when the CDROM is inserted into a running Windows system. For memory acquisition purposes a less invasive tool would be preferred. The actual duplicate could be stored on removable media or saved across a network. At a minimum, introducing new hardware such as a mass storage device would affect the registry, while creating a new network connection will create associated structures in RAM.

A second software technique involves the use a system crash. The notorious "blue screen of death" can occur under certain conditions outside of the control of the user, or it can be forced by the user. The user can force a crash either by using the built in CrashOnCtrlScroll [18] which requires a registry edit, or via a 3rd party utility such as NotMyFault.exe released by SysInternals (now owned by Microsoft). In either case, if the systems is configured to create a *FULL* crash dump (as opposed to *Mini, Kernel,* or *None* – which is controlled again by the registry) then the contents of memory will be eventually saved to a file. This save comes at the cost of losing the contents of the Pagefile, which when combined with the size of the subsequent file created upon reboot results in overwriting areas of the hard disk equal to or greater than twice the size of physical memory present in the system. This negative impact to non-volatile evidence through the changing of registry values, overwriting of unallocated space and potential for reboot[5] makes this method less preferred.

When using virtualization software, a virtual machine may be paused and the *virtual physical memory* (that is the abstraction of physical memory presented to the virtual machine) can simply be copied unbeknownst to the virtual machine. Of course, this software technique does not address the tangible physical RAM, and is mentioned primarily for completeness.

Hardware Acquisition

Hardware techniques are currently quite limited. Firewire has shown some merit for acquisition, because Direct Memory Access is possible via the IEEE 1394 specification, and proof of concept code has been released [19]. However, results of acquisition via Firewire vary widely. This technique not only has specific hardware requirements, but has also been shown to be inconsistent [20] and in some cases causes hardware to malfunction.

Hardware acquisition through dedicated hardware is the most desirable method. When using dedicated hardware the contents of RAM does not have to be altered in order to create the copy. This method currently has two very distinct drawbacks: it requires pre-meditation because the hardware must be in place prior to the incident, and there are no such products currently available to the consumer (but proof of concept has been created [21,22]). Arguably, this is the only technique that can *suspend* a typical (non-virtual) machine in order perform the acquisition.

Time Sliding Window

Since RAM is constantly in use, the contents of RAM are constantly changing. The amount of change varies greatly based on hardware, software, and usage of the system, but the fact remains that if the system is being used, RAM is changing. The fact that the contents are continuously changing paired with the necessity to acquire memory while the system

is running results in an inability to capture RAM at a precise point in time.

All of the above techniques[6] will exhibit a "time sliding-window" phenomenon where at least some portion of RAM was currently being altered at the time of the copy. Validation, such as an MD5 hash of original media before and of the duplicate after the copy, may work on unchanging stores like a hard disk drive, but one would expect it to not work on RAM (the contents of which are expected to have changed between hashing).

A case could be made for validating *similar* copies. Consider two RAM duplicates made as closely together as time allows, one created right after the other. Temporal proximity would suggest that "not much" had changed in the RAM contents between the two copy operations or at the very least that less change will have occurred than if the machine was left to run for extended or particularly busy periods. The amount of actual change could be quantified using a hash window equal to the page size. Pages that did not change between the two copy operations would have identical hashes, altered pages would have different hashes.

RAM ANALYSIS

Even if it is shown that creating a duplicate does have less negative impact to evidence than performing common incident response steps, the requirement for the information obtained during these steps still remains. The RAM duplicate serves little purpose without that ability to extract at least similar information that incident response tools can provide. Ideally, even more information can be garnered from the RAM duplicate.

Lack of Structure

Today most host based forensic analysis revolves around the inspection and interpretation of files and file systems. Recovering files, analyzing time stamps, file carving, etc typically all rely on file system specific concepts such as the File Allocation Table, Master File Table, inodes, and even clusters. This additional file system abstraction layer is not present when considering RAM. When compared to many types of files, much of the data in RAM may appear structureless.

The analysis of this *raw* data employs techniques from different areas such as kernel debugging and reverse engineering. In fact, in order to aide the analysis of the volatile data, often information from a non-volatile may be required. Consider employing a technique to find processes that is similar to using file headers for traditional file carving. Just as particular byte sequences such as 0xFFD8FFE0 or 0xFFD8FFE1 can be searched for at the beginning of a cluster on disk to identify possible JPEG headers, particular patterns can be sought at the beginning of a memory boundary (such as a page) in order to find possible structures such as a process. In the case of a JPEG the file format is well known in order to facilitate broad use of the file type. In the case of a process no format needs to be publicly available as the process structure was never intended to be disseminated to other systems. The lack of structure information is only compounded when considering closed-source operating systems. In order to seek out these structures, the format of the structure must be known prior to the search. A set of such structures can be calibrated using known systems. For example, through kernel debugging, it is readily apparent that the size and structure of a process differs between many Windows operating systems depending on version and service pack level (see Table 1: Windows Data Structure Offsets).

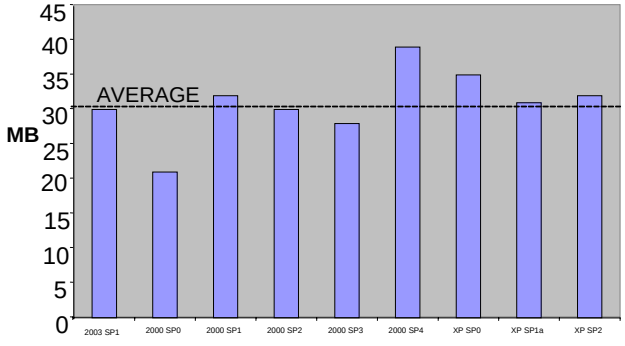
Table 1 : Windows Data Structure Offsets					
	2000	XP	XP SP2	2003	Vista
EP_PageDirBase	18	18	18	18	18
EP_processors	34	34	34	34	34
EP_T_Forward	50	50	50	50	50
EP_T_Back	54	54	54	54	54
EP_priority	62	62	62	62	64
EP_T_Quantum	63	63	6f	63	*
EP_T_Qant_dis	69	69	69	69	60*
EP_exitStatus	6c	24c	1d0	24c	234
EP_createTime	88	70	70	70	88
EP_exitTime	90	78	78	78	90
EP_PID(client Unique)	9c	84	84	84	9c
EP_WorkSetSize	e4	20c	20c	214	208
EP_WorkSetMin	e8	210	210	1f8	1ec
EP_WorkSetMax	ec	214	214	1fc	1f0
EP_AccessToken	12c	c8	c8	c8	e0
EP_PPID	1c8	14c	14c	128	124
EP_name	1fc	174	174	154	154
EP_size	290	258	260	278	268
TH_size	248	258	258	260	278
TH_createTime	1b0	1c0	1c0	1c8	1d0
TH_exitTime	1b8	1c8	1c8	1d0	1d8
TH_exitStatus	1c0	1d0	1d0	1d8	1e0
TH_PID (client unique)	1e0	1ec	1ec	1f4	1fc
TH_TID (client unique)	1e4	1f0	1f0	1f8	200
TH_isTerminated	224	248	248	250	250
TH_startAddr	230	224	224	22c	234

EP denotes the Windows EProcess structure, TH denotes EThread. All values are base 16 (hex).
*Quantum related values for Vista do not fit the pattern of prior OSes, and need to be researched further.
Values obtained via LiveKD and the Windows kernel debugger by issuing: dt -a -b -v _EPROCESS

Process Carving

As late as 2005, strings [23] analysis was considered the best method available to extract information from a RAM duplicate [24]. Running strings on a RAM duplicate acquired from a cleanly installed and booted Windows operating system resulted in the average extraction of more then 30 MB of largely unusable text[7] (see Figure 1: Strings found in a cleanly booted system). Keep in mind that these are unmodified operating systems, fairly atypical in the wild and that

Figure 1: Strings found in a cleanly booted system



even 30 MB of text would translate to roughly 8000 printed 8.5" x 11" sheets of paper.

A strings analysis will not be able to lend much insight about RAM specific structures such as processes. Instead, a search for known patterns must be performed along with a validation process for potential structures. The signature of a process can be defined by inspecting known offsets (as obtained from calibration) for expected data. For example, the offset related to process priority must be non-zero for all processes except the idle process, the offset related to the Page Directory Base (PDB) must be non-zero (a process must have a PDB) and the PDB must be on a page boundary (normally 4K), all the threads of a process must exist inside of the section of RAM dedicated to kernel memory, etc.

Assuming that the operating system version and service pack level are known prior to the search (i.e. obtained by inspection of the hard disk drive), a search for processes in a forensic image of 512 MB RAM takes about 7 minutes to execute through a PERL interpreter on a modest system[8]. This is a brute-force search that searches for structure signatures linearly. The fully commented proof of concept code is less than 1000 lines and a high success rate can be achieved implementing as few as five checks on known offsets. [25]

A handful of tools are now available for performing analysis similar to what is stated above. Among them are procloc [4], Windows Memory Forensic Toolkit [6], Windows IR tools [5], and memparser [19] which was one of the original DFRWS submissions that was later released publicly. Each of these tools have their various benefits and drawbacks, mostly associated with project maturity. For example, many tools do not have a good user interface and many only work on RAM from some versions of Windows. As with other tools, be sure to adequately test these tools before using any of them in a non-academic sense.

TRUST ISSUES

Issues with trust arise in both the acquisition and analysis phases. The most detrimental issue involves the acquisition of RAM contents. This situation revolves around the problem of executing code on an untrusted system. How can one be assured that the input to the copy operation is actually the contents of the systems RAM? Techniques could be employed by malware to deny access to RAM or worse, to misrepresent the contents of RAM in order to elude detection. Many rootkits already use similar techniques. However, in a situation involving such malware one could easily make the argument that this misrepresentation would also affect common incident response tools. It is currently thought that the only way to completely mitigate this threat is to use dedicated hardware for acquisition[9].

Trust is also an issue during analysis. For example, some of the above tools make use of assumptions about internal Windows process scheduling. Windows maintains a doubly linked list of process structures, each process structure contains information on where the next and previous process structures are located. If this information is trusted, it greatly speeds up the enumeration of processes. However, if a process (thread) has become unlinked from this list it will not appear in the enumerated set. This could be the case for processes that are no longer scheduled for execution (old processes) as well as hidden processes.

A final trust issue is foundational to a core computer science concept: RAM may not be as volatile as one might have thought. It has been demonstrated that the contents of RAM can actually survive reboots and even short durations of power completely removed from a system [27]. This actually challenges term "volatile memory." Computing systems can not be trusted to provide RAM in a *clean* state initially, only an *unknown* state. Further research must be performed in order to determine if this known ledge can be leveraged in the favor of a responder.

FUTURE WORK

For most purposes, the area of memory forensics can be considered to be less than 2 years old, still in its infancy. As with other budding areas of research memory forensics is ripe with possibilities for both unique research and refinement of existing research. Below find suggestions for new research in addition to ideas on how to extend upon the concepts provided here.

Compare the trusted process list with one obtained via brute force methods. A brute force technique was described in this article. Others [6] use a list traversal approach. Comparing results from the two methods could flag outliers, such as hidden processes.

One could employ virtual memory unification. Since the RAM duplicate being analyzed is never actually executed by the CPU, it does not have to obey typical memory management

rules, such as those related to paging. For example, during analysis all pages could be "swapped in" from the pagefile extracted from a forensic duplicate of the hard disk drive.

Operating system detection could be improved. The execution time mentioned for a brute force search assumes that the operating system version and service pack level are known. If this information is not known (you have a RAM image but no hard disk drive image, or an encrypted hard disk drive) then the best case is to try all known operating system offsets until one search provides enough results to be deemed correct. This increases the execution time 1 factor for every known operating system. For example, instead of 7 minutes, procloc could take 35 minutes to execute.

One could automate the correlation with non-volatile stores. It was mentioned above that some information from a hard disk drive is very useful in the analysis of RAM. Operating system type and service pack level for example. Other types of information are also very valuable. Consider the need to link a process to a user account. The process structure only stores the internal UID which must then be correlated with information in registry to obtain a username.

Executables could be automatically or selectively extracted from the RAM duplicate. Assuming that outliers could be easily identified (as suggested by list comparison earlier), executables could automatically be created from the extracted information in order to automate analysis.

Flag rogue structures by employing more checks. It was shown in this article that accurate results could be achieved with as few as 5 checks. Malware that is "aware" of these checks could attempt to spoof them in order to "fit in." Employing more checks and adjusting their strictness could in effect identify structures with varying levels of "correctness."

Account for all areas of memory by marking sections as structures are found. Consider a mature field of memory forensics, where processes, threads, file caches, etc. all have reliable tools that allow inspection and extractions. If each of these tools marked the areas of memory that it found to be a legitimate structure, then what do the unmarked areas represent? This technique would be similar to code coverage procedures use in other discipline.

Most current tools only support environments that are either easy to develop tools for, or represent a large user base. Future tools need to support fringe memory architectures such as those enabled by the /PAE and /3G boot switches, non-i386 support is needed, and of course tools need to keep up with current operating systems and add support as needed (such as Vista).

CONCLUSION

In exchange for a minimal negative impact (potentially as small as creating a single new process) to evidence during acquisition, a much greater depth and breadth of information concerning system state can be gained during analysis. The ability to gather pertinent information from a RAM duplication often requires information to be gathered from a related non-volatile store prior to analysis, but may require little acquisition training and minimal additional hardware. At the very least, RAM acquisition allows analysis to occur after first response and enables RAM data to be viewed as an additional static evidence item to which traditional preservation and duplicate validation techniques can be applied.

NOTES

1. Guidance software has sections of their website (www.guiadancesoftware.com) devoted to e-discovery using their EnCase product line. Additionally there are many conference presentations and whitepapers on the subject, but no traditionally academic sources. (e.g. CSI Annual Computer Security Conference, CEIC, DoD Cyber Crime Conference)dnotes will go here when I figure out how to do this in Word.

2. /3GB and /PAE are options given at boot time for MS Windows based operating systems that alter the default behavior of memory. Physical Address Extension (PAE) is heavily, if not completely, related to Intel IA-32 architecture PAE (Pentium Pro and above) basically increases physical addresses to more than 32 bits. 3GB allows for applications to use 3 GB of virtual address space instead of the normal 2 GB. [29,30]

3. The command should be typed all on one line, not two lines as shown. Notice the specified size of 4K which corresponds to the size of a memory page. Note that usermode access to the PhysicalMemory object has been removed by Microsoft in Windows Server 2003 SP1 and potentially in future operating systems. \.\DebugMemory is being researched further.

4. It could be argued that this information is not lost, but will likely be swapped out. This would depend if the portion of memory in question as allocated or not, and even if the portion was allocated that subsequently swapped out, some information in the swap file would be lost.

5. A crafty approach would be to invoke the crash dump which writes physical memory contents to the physical sectors of the hard disk where the pagefile is stored. Then unplug the system after the dump is complete but before POST. In this situation the contents of the pagefile are still lost, but the dump is not written as a file to the file system and the system did not actually reboot (changing timestamps and similar). Using a write blocker the RAM contents could be extracted from a forensic duplicate in order to perform RAM analysis.

6. This may prove to not be the case with a dedicated hardware acquisition, but this cannot be tested as no such hardware readily exists.

7. Tested on systems with 512 MB of RAM.

8. Tested on a IBM Thinkpad R51, with 1.5 Ghz Intel Pentium 3m with 1 GB of RAM, running Windows XP SP2 and ActivePERL 5.8.7.

9. Which remains to be seen, not only is such hardware not yet available, but circumvention of such hardware have already been claimed [28]

REFERENCES

1. DEPARTMENT OF DEFENSE TRUSTED COMPUTER SYSTEM EVALUATION CRITERIA (TCSEC) DOD 5200.28-STD. US Department of Defense. December 1985.

2. United States Secret Service. Best Practices for Seizing Electronic Evidence. Second Edition. 2002.

3. Nolan, O'Sullivan, Branson, Waits. First Responders Guide to Computer Forensics. Carnegie Mellon University 2005.

4. Tim Vidas. Procloc. http://nucia.unomaha.edu/tvidas/. Accessed Feb 8, 2007.

5. Harlan Carvey. Windows IR/CF Tools. http://sourceforge.net/projects/windowsir/. Accessed Feb 8, 2007.

6. Mariusz Burdach. Windows Memory Forensic Toolkit. http://forensic.secure.net. Accessed Feb. 8, 2007.

7. Andreas Schuster. PTFinder. http://computer.forensikblog.de/en/. Accessed Feb 8, 2007.

8. McAfee VIL database. OneHalf virus. Accessed Feb. 8, 2007. http://us.mcafee.com/virusInfo/default.asp?id=alpha

9. McAfee VIL database. Nimda worm. Accessed Feb. 8, 2007. http://us.mcafee.com/virusInfo/default.asp?id=alpha

10. McAfee VIL database. SQLslammer worm. Accessed Feb. 8, 2007. http://us.mcafee.com/virusInfo/default.asp?id=alpha

11. Goodwin, Bill. High-tech crime is put on trial. ComputerWeekly.com. Jan 27, 2007. Accessed

Apr 30, 2007. http://www.computerweekly.com/Articles/2007/01/27/221526/high-tech-crime-is-put-on-trial.htm

12. United States vs O'Keefe. D.C. Docket No. 04-0001 Cr-WLS-1. No 05-11924. Georgia App. Ct. Aug 22. 2006. http://www.ca11.uscourts.gov/opinions/ops/200511924.pdf

13. St. of AZ vs Brandy. S-0700-CR-2005014635. Arizona Sup. Ct. Nov. 11 2005.

14. Auditor Acquitted – Uses Computer Virus Defense. Aug 28 2003. Accessed Mar 28, 2007. http://www.accountingweb.com/cgi-bin/item.cgi?id=98024

15. United States vs Michael Shawn McCourt. District Court for the western district of Missouri. 06-1018. Nov 24, 2006. http://www.ca8.uscourts.gov/opndir/06/11/061018P.pdf

16. Matthew David Bounds v The Queen. HCA 39. July 20, 2006. http://www.austlii.edu.au/au/cases/cth/high_ct/2006/39.html

17. Altheide, Cory. Forensic analysis of Windows hosts using UNIX-based tools. Journal of Digital Investigation. Vol 1, Num 1. Feb 2004.

18. KB 244139: Windows feature allows a Memory dump file to be generated with the keyboard http://support.microsoft.com/kb/244139/en-us

19. Adam Boileau. Hit By A Bus: Physical Access Attacks With Firewire. Ruxcon 2006

20. GM Garner. Memory image differences in Firewire acquisition. http://www.storm.net.nz/projects/16

21. Carrier, Grand. A hardware-based memory acquisition procedure for digital investigations. Digital Investigation Journal. Issue 1, p 50-60. Feb 2004.

22. Petroni, Fraser, Molina and Arbaugh. Copilot – a Coprocessor-based Kernel Runtime Integrity Monitor. Proceedings of the 13th USENIX Security Symposium. Aug 9-13, 2004.

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

24. Stover S., Dickerson M. Using Memory Dumps in Digital Forensics. ;Login: magazine. Volume 30, Issue 6. December 2005.

25. Vidas, Timothy. Starting a Framework for the Analysis of Volatile Data Stores. Third Annual ifip WG 11.9 International Conference on Digital Forensics. Orlando, Florida. Jan 28-31, 2007.

26. Chris Betz. Memparser. http://sourceforge.net/projects/memparser/. Accessed Feb 8, 2007.

27. Chow, Pfaff, Garfinkel, Rosenblum. Shredding Your Garbage: Reducing Data Lifetime Through Secure Deallocation. 14th USENIX Security Symposium. July / August 2005.

28. Joanna Rutkowska. Beyond the CPU: Defeating Hardware Based RAM Acquisition Tools. Will be given at Black Hat DC 2007. http://blackhat.com/bh-dc-07/bh-dc-07-speakers.html#Rutkowska. Accessed Feb 8, 2007.

29. Memory Support and Windows Operating Systems. Feb 9, 2005. Accessed March 28, 2007. http://www.microsoft.com/whdc/system/platform/server/PAE/PAEmem.mspx

30. Intel 64 and IA-32 Architectures Software Developer's Manual: Volume 3A: System Programming Guide Part 1. Intel Corp. November 2006.