

7-1-2006

Acquisition and Forensic Analysis of Volatile Data Stores

Timothy Vidas
University of Nebraska

Follow this and additional works at: <https://digitalcommons.unomaha.edu/studentwork>

Recommended Citation

Vidas, Timothy, "Acquisition and Forensic Analysis of Volatile Data Stores" (2006). *Student Work*. 2167.
<https://digitalcommons.unomaha.edu/studentwork/2167>

This Thesis is brought to you for free and open access by DigitalCommons@UNO. It has been accepted for inclusion in Student Work by an authorized administrator of DigitalCommons@UNO. For more information, please contact unodigitalcommons@unomaha.edu.



Acquisition and Forensic Analysis of Volatile Data Stores

A Thesis

Presented to the

Department of Computer Science

and the

Faculty of the Graduate College

University of Nebraska

In partial fulfillment

of the Requirements for the Degree

Master of Computer Science

by

Timothy Vidas

July 2006

UMI Number: EP73709

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



UMI EP73709

Published by ProQuest LLC (2015). Copyright in the Dissertation held by the Author.

Microform Edition © ProQuest LLC.

All rights reserved. This work is protected against
unauthorized copying under Title 17, United States Code

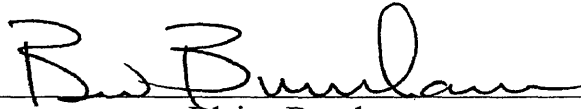


ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

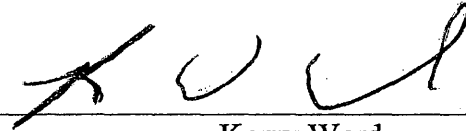
THESIS ACCEPTANCE

Acceptance for the faculty of the Graduate College,
University of Nebraska, in partial fulfillment of the
Requirements for the degree Master of Science in Computer Science,
University of Nebraska at Omaha.

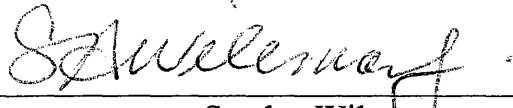
Committee:



Blaine Burnham

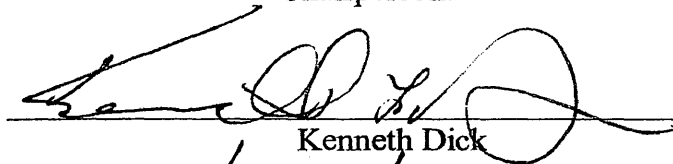


Kerry Ward

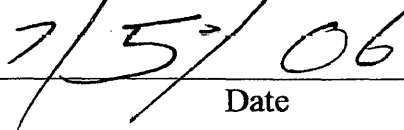


Stanley Wileman

Chairperson:



Kenneth Dick



Date

Acquisition and Forensic Analysis of Volatile Data Stores

Timothy Vidas, MS

University of Nebraska, 2006

Advisor: Kenneth Dick

The advent of more witted threats against typical computer systems demonstrates a need for forensic analysis of memory-resident data in addition to the conventional static analysis common today.

Some tools are starting to become available to duplicate various types of volatile data stores. Once the data store has been duplicated, current forensic procedures have no vector for extrapolating further information from the duplicate. This thesis 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, while creating as small an impact as possible to the state of a system.

It is intended that this thesis will give insight to obtaining more post incident response information along with a smaller impact to potential evidence when compared to typical incident response procedures.

Table of Contents

| | |
|---|-----|
| THESIS ACCEPTANCE..... | 2 |
| Abstract..... | 3 |
| Table of Contents..... | 4 |
| List of Tables..... | 5 |
| List of Figures..... | 5 |
| Assumptions:..... | 6 |
| Disclaimer:..... | 7 |
| Definitions..... | 8 |
| Abbreviations..... | 12 |
| The Forensic Process..... | 14 |
| Background..... | 16 |
| Virtual Addressing and Paging..... | 17 |
| Ringed Architecture..... | 19 |
| Processes and Threads..... | 20 |
| Objectives..... | 21 |
| Memory Acquisition..... | 31 |
| Memory Analysis..... | 44 |
| Original Analysis Goals:..... | 45 |
| The Windows EPROCESS structure and significance..... | 46 |
| Finding processes in memory image..... | 50 |
| Conclusions..... | 52 |
| Direction..... | 53 |
| Appendix A: Methodology notes..... | 55 |
| Appendix B: EPROCESS dumps of a live typical system – XP SP2..... | 56 |
| Appendix C: EPROCESS structure of Microsoft Windows 2000, Service Pack 4..... | 65 |
| Appendix D: ETHREAD structure dump from WinXP SP2..... | 71 |
| Appendix E: Decoding a Process Owner..... | 76 |
| Appendix F: Offset Deltas by service pack..... | 80 |
| Appendix G: Proof of Concept Source Code Listing..... | 81 |
| Appendix H: Sample runs..... | 99 |
| Appendix I: Linux Acquisition..... | 105 |
| Appendix J: GNU General Public License..... | 112 |
| Cited Works..... | 120 |

List of Tables

| | |
|--|----|
| Table 1: Common Incident Response Steps | 30 |
| Table 2: : Windows pagefile sizing by Architecture | 33 |
| Table 3: Maximum RAM support by OS | 33 |
| Table 4: EPROCESS Structure..... | 47 |
| Table 5 : SID Encoding..... | 79 |
| Table 6 : Windows Data Structure Offsets | 80 |

List of Figures

| | |
|---|----|
| Figure 1: Digital Forensics Communities | 14 |
| Figure 2: Virtual Address Translation | 17 |

Assumptions:

Only host based forensics are discussed in this thesis. Though interesting tangents may be heavily related to the work shown here (such as the volatile stores of a network switch) the omission of network based forensics is intended.

All of the samples and some of the text assumes standard Intel x86 32-bit architecture and while many things may be similar, many alterations would likely have to be made for 64-bit platforms.

In most cases the term “non-volatile store” refers to a technology such a Hard Disk Drive (HDD) which is assumed to retain data over extended periods of time with no power applied to the device. Similarly, the term “volatile store” refers to a technology such a Random Access Memory (RAM) which is assumed to not retain data over extended periods of time with no power applied to the device. Various kinds of copies of these types of stores may be referred to as duplicates, copies, or images.

Disclaimer:

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-discovery¹ 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 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; completeness versus speed and storage advantages respectively. This text does not debate these techniques.

When copying Random Access Memory (RAM) from a live system the contents will change as the copy is being created. This not only makes validation of the copy difficult, but also questions the very terms used to describe this copy. *Duplicate*, *Image* and even *copy* may not be the best suited terms, but are used here in the absence of a better term.

¹ Guidance software has sections of their website (www.guidancesoftware.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)

Definitions

`%SystemRoot%`: A Windows™ environmental variable is denoted by percent symbols on either side. SystemRoot refers to the directory in which the OS is located in, typically `c:\windows` or `c:\winnt`.

Binary: The low-level form of an application; it is typically executable, not readily readable by a human, and not portable between platforms. Source Code is compiled in order to create an executable binary.

Boot volume: The volume that contains the operating system and its support files. In Windows, the boot volume can be, but does not have to be, the same as the system volume.

Closed source: Closed source software is software for which the source code is not open to public view. Under most licenses users cannot modify such software or redistribute it. Typically this software is distributed in pre-compiled (binary) form.

Computer Forensics: The application of computer science to questions which are of interest to the legal system

Digital Forensics: The use of scientifically derived and proven methods toward the preservation, collection, validation, identification, analysis, interpretation, documentation, and presentation of digital evidence derived from digital sources for the purpose of facilitation or furthering the reconstruction of events which may be found to

be criminal, or helping to anticipate unauthorized actions shown to be disruptive to planned operations.

Driver : A device-specific program that enables a computer to work with a particular piece of hardware, such as a printer, disk drive, or network adapter. Because the driver handles device-specific features, the operating system is freed from the burden of having to understand—and support—the needs of individual hardware devices.

Forensic Duplicate: Commonly described as a “bitstream” or “bit for bit” copy of a hard disk. More accurately it is a sector by sector copy from source media. It is stored in a ‘raw’ unaltered form.

Forensics: The application of a science to questions which are of interest to the legal system.

Image: As a noun, refers to some form of Forensic Duplicate. As a verb, refers to the process of creating a Forensic Duplicate.

Incident Response: The practice of detecting a problem, determining its cause, minimizing the damage it causes, resolving the problem, and documenting each step of the response for future reference.

Kernel: The fundamental part of an operating system. It is a piece of software responsible for providing secure access to the machine's hardware to various computer programs. Since there are many programs, and access to the hardware is limited, the kernel is also

responsible for deciding when and how long a program should be able to make use of a piece of hardware.

LiveCD: Operating System stored on a Bootable CD. It does not require a hard drive in order to execute. A virtual disk is typically created in RAM in order to facilitate programs that require a file system in order to operate.

Media Analysis: The use of procedures similar to those used in Computer or Digital Forensics, but with no intent of involvement of a legal system.

Open Source: A movement in the programming community for making source code (program instructions) free and freely available to anyone interested in using or working with it. Such source code may be distributed only as uncompiled source, but in many cases also includes compiled (binary) versions for ease of use by the end user.

Postmortem: Discussion of an event after it has occurred: literally, occurring after death.

Process: The state of a program when execution is actually occurring on a machine along with the context required to execute.

Source Code: The human readable form of software. It is typically written in a high-level language such as C++. Machines cannot readily execute code in this form; it must first be converted to a low-level form, typically through a process called compilation.

STOP code: The error code that identifies the error that stopped the system kernel from running.

System Volume: The volume that contains the hardware-specific files that are required to load Windows. The system volume can be, but does not have to be, the same as the boot volume. `Boot.ini`, `Ntdetect.com`, and `Ntbootdd.sys` are examples of files that are located on the system volume.

Thread: A processor activity in a process. The same process can have multiple threads. Those threads share the process address space and can therefore share data.

Qualified Forensic Duplicate: Similar to a Forensic Duplicate, but stored in an altered form (e.g. compressed) or with the addition of some metadata. The process can be reversed or otherwise shown to accurately reflect the same data as a Forensic Duplicate.

Abbreviations

| | |
|-------|---|
| ARP | Address Resolution Protocol |
| BIOS | Basic Input Output System |
| BSD | Berkeley Software Distribution |
| CD | Compact Disk |
| CDROM | Compact Disk (Read Only Media) |
| CPU | Central Processing Unit |
| DD | Data Duplicator |
| ELF | Executable and Linking Format |
| EXE | Executable file (Windows) |
| GB | Giga Byte |
| GNU | GNU's not Unix |
| GPL | GNU Public License |
| HDD | Hard Disk Drive |
| IBM | International Business Machines™ |
| KB | Kilobyte |
| KVM | Keyboard Video Mouse |
| LE | Law Enforcement |
| MB | Megabyte |
| MD5 | Message Digest (version 5) |
| NUCIA | Nebraska University Consortium on Information Assurance |
| NX | No eXecute |
| OS | Operating System |
| PAE | Physical Address Extension |
| PCB | Process Control Block |
| PDA | Personal Digital Assistant |
| PDI | Page Directory Index |
| PDB | Page Directory Base |
| PEB | Process Environment Block |
| POFF | Page Offset |
| PTE | Page Table Entry |
| PTI | Page Table Index |
| RAM | Random Access Memory |
| RFC | Request For Comment |
| SID | Security Identifier |
| SMSS | Session Manager Subsystem |

| | |
|-------|---|
| SP | Service Pack |
| STEAL | Security Technology Education and Analysis Laboratory |
| TB | Terabyte |
| TLB | Translation Lookaside Buffer |
| US | United States |
| | |
| USB | Universal Serial Bus |
| VM | Virtual Machine |
| XD | eXecute Disabled |

The Forensic Process

The term *forensics* has many meanings. Alone, the word is defined by Merriam Webster as “relating to or dealing with the application of scientific knowledge to legal problems” (Forensic. Merriam-Webster). In the digital arena, however, many actions that bear resemblance to procedures used in the forensic process (media analysis, data recovery, event reconstruction and similar) are often billed as forensic services even though there is never any intention of applying the science to a legal problem. When computer science is applied to a legal process it is known as Computer Forensics (or depending on context, Cyber Forensics or Digital Forensics).

Whereas computer forensics is defined as “the collection of techniques and tools used to find evidence in a computer”, digital forensics has been defined as “the use of scientifically derived and proven methods toward the preservation, collection, validation, identification, analysis, interpretation, documentation, and presentation of digital evidence derived from digital sources for the purpose of facilitation or furthering the reconstruction of events found to be criminal, or helping to anticipate unauthorized actions shown to be disruptive to planned operations” (Reith, Carr, Gunsch. 2002.)

The basis of these steps traditionally revolve around preserving a state of a computer system for subsequent analysis and reporting. This analysis is commonly performed in parallel by more than one party, such as two sides in a legal dispute, and care must be taken to ensure that all parties involved are working with identical data. Nearly identical

or similar data is insufficient. It has been shown that even when working with large sets of data, minute a discrepancy can have profound effect². As such, concepts such as chain-of-custody borrowed from other disciplines are often adapted to digital forensics.

Even so, the Digital Forensics Research WorkShop (DFRWS) uses the Venn diagram shown in Figure 1 (Marc Rodgers, 2004)

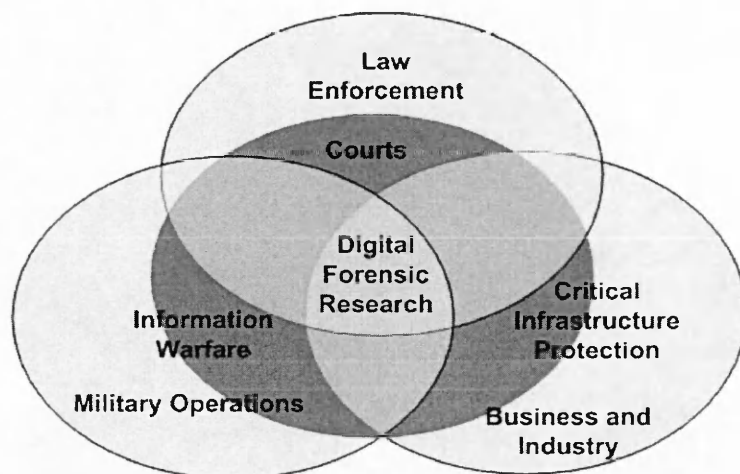


Figure 1: Digital Forensics Communities

to demonstrate three distinct communities of digital forensics. Only one community regularly deals with legal process in the course of performing digital forensics. It has been suggested (Rodgers, 2004, among others) that academia be added as a fourth community, further minimizing the root meaning of forensics.

² An example of a minute discrepancy having a profound effect is a single ASCII character (likely 8 bits) found in a document on a 80 gigabyte hard drive, becoming the deciding factor of a multimillion dollar settlement (about 1 / 85,000,000,000th of the total data). (Taub, 2006)

Background

In the not-too-distant past, a common incident response step taken early in the process was to ‘pull the plug’ on a powered on machine (United States Secret Service, 2002).

Investigators knew that performing a ‘clean’ shutdown could further change the state of the system. However ‘pulling the plug’ also has its own drawbacks to later analysis. One such drawback is the lack of ability to identify and examine the state of the machine at the time of seizure.

Some tools³ allow the acquisition of the contents of ‘raw’ RAM from a running system.

Thus far, the analysis of a RAM image has been limited to small special-use devices such as Palm PDAs or various cellular phones. For most forensic cases seen today, traditional post-mortem techniques are 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.

Many times the state of the non-volatile devices, such as hard disks, depend upon the state of a volatile device, such as RAM. This is the case with many forms of Hard Disk Drive encryption, where if a disk is powered down a secondary connection to the device

³ Helix Live CD – Incident Response Toolkit <http://www.e-fense.com/helix/>
Paraban’s Cell Seizure http://www.paraben-forensics.com/cell_models.html , etc.

may prove fruitless.⁴ In certain cases a multi-partite memory resident virus⁵ can even partially encrypt portions of a drive without the consent (or knowledge) of the user. In such a case, capturing and analyzing the contents of memory would be required for an investigation.

Virtual Addressing and Paging

In order to allow each process to have a logically contiguous address space and preserve the efficiency of not having to allocate contiguous memory addresses to each process, most modern OS memory management systems employ virtual addressing. In this type of addressing, processes are given virtual addresses for memory which are then translated to the correct physical address by the memory management system. This should not be confused with Windows Virtual Memory which enables a process to use more memory than physically available by swapping portions of memory to a secondary store such as a file on disk. In fact, in Windows, all OS instances are given a virtual address space of 4GB regardless of physical RAM installed. This 4GB virtual range is typically divided into two 2 GB sections⁶ – one for the OS and one for private application space. In this sense “RAM is a limited resource, whereas virtual memory is, for most practical purposes, unlimited.” (KB 555223).

⁴ It is very common in typical procedures to power off a system, then attach a write-blocking device to the HDD before connecting it to some other device independent of the original suspect hardware for acquisition purposes.

⁵ ONE-HALF virus <http://vil.nai.com/vil/content/Print98226.htm>

⁶ Baring some boot switches such as / 3GB.

When memory use exceeds the available RAM, portions of memory are typically paged (or copied) out to disk. In Windows this is done in 4 KB⁷ *pages* to files called the *Pagefile* (pagefile.sys). When data in a particular page is needed for processing the page is paged back into RAM and another page copied to the pagefile. For the purpose of this thesis, concepts such as reserving, locking, sharing and committing pages, will not be discussed, nor will the application of rights to pages via hardware memory protection⁸.

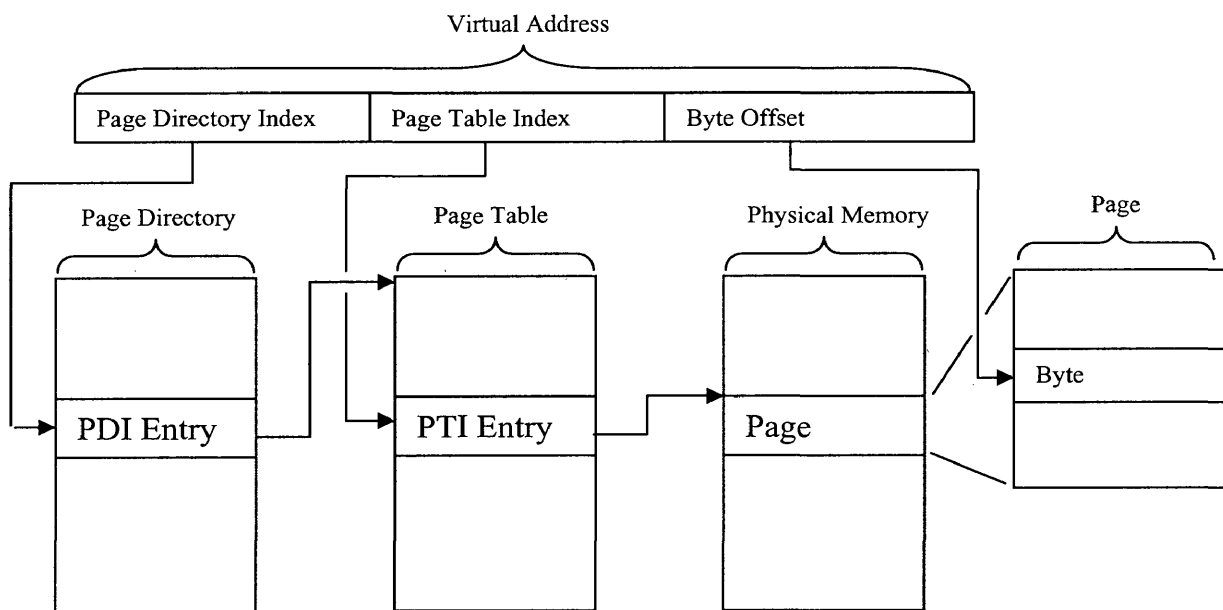


Figure 2: Virtual Address Translation

A memory manager generally constructs *page tables* to facilitate the translation of an object at a virtual address referenced in a process to the physical hardware location of that object. Each virtual address has an associated Page Table Entry (PTE) in the table

⁷ In X86 without the PAE switch enabled. For x64 and IA64 page size will range from 4 KB to 16 MB. The cost vs benefit is typically related to Translation Look Aside Buffer (TLB) efficiency in hardware and will not be discussed further here.

⁸ Since the page is the smallest granularity assignable via hardware memory protection, some concepts have been getting attention again lately such as AMD's NX bit or Intel's XD bit (bit 63 of the page table entry).

which contains the physical address. In the Intel x86 architecture, a Page Directory Index (PDI) also exists in order to locate the correct Page Table in which a PTE exists. When a process requires access to a byte at a given virtual address (also see Figure 2: Virtual Address Translation):

- the PDI is referenced to locate the correct page table
- the corresponding PTE found in the page table for the page that contains the byte in question, contains the physical location of the page in RAM
- finally an offset into the page is used to locate the actual physical byte

An example of decoding a virtual address can be found in Appendix E: Decoding a Process Owner. It is important to note that each Windows process has a single page directory with 1024 entries, and up to 512 page tables, each with 1024 entries. Paging is typically enabled very early in the boot process, and applications benefit from virtual memory / paging without any alteration because it is provided by the OS, essentially at a lower layer than the process.

Ringed Architecture

Rings (aka protection rings, processor modes, process privilege mode), are provided by many processors to allow “memory access protection from two levels (user and kernel).” (Hennessy, Patterson. 2003). A system requires at least two protection levels to provide privilege isolation between processes, which becomes a foundational requirement to provide notionally higher concepts such as file confidentiality (Ware. 1970). Kernel (or supervisor) mode allows access to all CPU functionality while the non-kernel modes

(typically only one: user mode) allow restricted access. If the architecture allows more than two modes, and the modes are implemented in a singular hierarchy, the architecture is ringed; if the architecture provides only two modes, it can be thought of as a 2-ring architecture. Generally, processes that exist in rings ‘further away’ from the kernel mode (typically increasing in number) the functionality available to the process in that ring is a subset of the prior ring.⁹ Most popular processors support a certain number of rings (Intel x86 supports 4), however most operating systems only implement two rings for kernel and user modes (typically at ring 0 and 3 respectively) and provide further protection granularity through OS features instead of rings. This distinction between the two rings is important to note for a variety of reasons, among them are the following. Processes in ring 3 may not be able to access data in ring 0 and this may hinder the acquisition process. Similarly rogue processes in ring 0 may not be detectable by processes in ring 3. Among other examples, this distinction between the two rings can make kernel level rootkits difficult to detect and/or remove. (Hoglund, Butler. 2006)

Processes and Threads

A process is the state of a program when execution is actually occurring on a machine along with the context required to execute: current values of the program counter, registers and variables. (Tanenbaum, 1997) A Windows process consists of a private virtual address space, the actual executable code, a list of open file handles, a certain

⁹ Correct implementation of a ringed architecture requires both hardware and software (OS) support. The concept of ringed architecture has existed for quite some time. The Multics project supported ringed architecture circa 1963 (Corbato, Vyssotsky. 1965).

security context, an ID, and at least one thread. (Russeinovich, 2005). It is important to note that even in the absence of a multithreaded program, or even the possibility of allowing multiple threads, every process has at least one thread. Only a thread can execute, which counters popular terminology related to “running processes.”

Many processes are deemed *default*, and some are deemed *required* by various sources. Default processes are those that start with the booting of a typical installation of an OS. Required processes are those that are required for the OS to function. Processes may have familiar relationships such as parent and child, where the parent process starts the child process and so on. Most operating systems are distributed with applications that facilitate viewing the state of currently running processes such as the Windows Task Manager or the Linux `ps` or `top` commands. Using these tools, different information about the processes may be studied. The internal structure of a process and its respective thread(s)¹⁰ are paramount to the analysis portion of this text and a more detailed summary of related data structures will be presented there.

Objectives

The goal of this work is to assess the current methods and mechanisms available to duplicate volatile data stores and more so, to analyze the effects these tools have on the state of the system in which these stores exist. Some (most, all?) tools will actually alter

¹⁰ It is important to point out that this thesis only refers to “full” threads and not “lightweight” threads or *fibers* which are scheduled internal to a process and not by the OS scheduling routine. These are obviously very specific to each application and are not discussed here.

the state of the system in question which is not recommended from a forensic point of view. Altering the state of the system is akin to modifying a physical crime scene, the evidence may not be altered, but there is no way to know after the modification has occurred. The goal of the media analyst is often to glean as much information as possible directly from the evidence; the goal of a party involved with a legal system may fall more in line with concepts such as burden of proof which allows for inference in many situations.

In physical forensics, malfeasance or misconduct that changes the state of a crime scene could quite likely render evidence unusable in the court system. Digitally, the court is taking a similar approach, however the circumstances are not equivalent. For example: if a murder has occurred in a kitchen, and the murder weapon was left at the scene, if the kitchen is sealed and guarded the murder weapon will still exist at a later point in time. If a computer crime is observed, evidence might well be lost over time due to the normal operation of a computer system. Different, common actions on an individual system such as scanning, paging, defragmenting, and re-allocation of clusters/blocks can all alter or overwrite potential evidence on disk. These actions (and others that do not necessarily affect the disk) may start new processes and utilize portions of memory which may alter or overwrite potential evidence in RAM. The problem of lost potential evidence may even be compounded by common circumstances like the active participation of the system in question on a network. Generally, the more active a system the more likely it is that potential evidence will be lost.

Regardless of the effectiveness of the methods and mechanisms used for acquisition, procedures will be created to perform incident response and media analysis akin to those in use today for traditional media. This thesis focuses on the analysis of the different portions of RAM used by mainstream operating systems in order to adapt current response methodologies to further preserve the state of a suspect system. It is intended that the effect of current incident response procedures on a suspect system be lessened and the amount of information available after the initial response be equal to or greater than the information obtained using current procedures.

Scripts developed to aid in the analysis of an acquired image of a volatile data store are distributed open source as Appendix G: Proof of Concept Source Code Listing to this thesis under the GNU General Public License (Appendix J: GNU General Public License) for public use. However, the scripts are not the primary focus here.

The analysis of volatile stores and traditional postmortem forensics vary greatly. Traditional forensics typically involves the postmortem media analysis of a file system. Though it is common to speak of analyzing a particular workstation or personal computer, the analysis is very often only performed 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” (a “pseudo” file system) 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 may allow an analyst to quickly view all digital pictures, 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 misnomer “bit-for-bit” duplicates of hard disks¹¹ are parsed and data that was not contained within the suspect file system on the original disk is added to the container in the analysis software (such as deleted files). Thus traditional analysis depends very heavily on the understanding of the file system that was used on the original 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), other factors become main focuses: the Operating System in use, the configuration of that Operating System and possibly other information such as hardware implementation.

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

¹¹ Most duplicates that are represented as bit-for-bit or byte-for-byte duplicates are actually sector-by-sector duplicates as the hard disk controllers on modern hard disk drives typically are only capable of providing data at a sector granularity

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. Contrary to popular belief, data may still exist in a volatile data store from a time prior to the last reboot of the system (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.

Much information either required for or beneficial to the analysis of volatile stores will likely only be attainable from the non-volatile stores. While many configuration settings are standard, there is no technical OS control preventing something like a non-standard page size. In some cases, changing these settings may actually be recommended¹² (Marxmeir, 2001). Data structures and memory mapping often differ between different releases of software. Therefore obtaining information from a non-volatile store from the suspect system version (like the version of OS from the hard disk) can be quite beneficial for the analysis of the volatile store (such as parsing processes from data structures in RAM. Cisco’s IOS router software alters the mapping of memory in every software release (Lynn, 2005).

¹² While this reference does not speak directly to the topic at hand, “...the file system block size could have a big impact on the system performance...” as related to databases. A search the a field of choice should find many articles both by vendors and end users relating storage unit sizes of all kinds to system performance. Page size can also be quite a bit different in non x86 architectures.

Only recently has the capture of certain non-volatile stores become automated enough to have the potential to be a common incident response action (Helix Version 1.4, 2004). A few contemporary tools have built-in functionality for imaging RAM, however, once captured, the customary analysis of this image is done manually using a hexadecimal editor/viewer and possibly some slightly more helpful, yet still primarily crude techniques such as performing a strings analysis. The problems are in the tools and techniques provided to the typical forensic analyst, and their focus on non-volatile stores.

This does not even touch on the present-day debate on whether is advisable to alter the state of a currently powered on system in exchange for obtaining more information. Historically, a first responder was trained to “pull the plug” if a suspect machine was discovered in a powered on state¹³ (United States Secret Service, 2002). This typically does guarantee ‘more’ information to be available on the non-volatile stores because the system has not had the chance to perform any shutdown tasks. For example, an operating system may clear a paging file, or delete temporary files at shutdown. However, this approach will have marginal success at showing the entire state of the system at the point that the power was removed because the OS has not been given the opportunity to perform shutdown tasks.

Many texts have proposed that incident response should observe an order of volatility, such as processing stores in a particular order: registers, routing information, process

¹³ “...if a specialist is not available...disconnect all power sources; unplug from wall and the back of the computer.” This verbiage is from the US Secret Service, but the action is not atypical.

table, temporary files, physical disk (Brezinski, Killalea. 2002). Each type of store respectively becomes less volatile and more persistent as the process goes on, and in fact some first responder checklists may employ executing certain commands or scripts before removing power from a system. These procedures may include running commands to obtain information about the state of the system, such as process lists, network connection status, open files, etc. The perceived primary problem with this procedure is that potential evidence is being altered; many incident response guides do not even take this state alteration into account (Baker, 2005 among others). Simply performing the response procedures introduces more processes in the process table of the machine. Of course, these processes will be bound to the access level that they run under. Not having administrator level access at incident response time, or the presence of a rootkit or other subversion technique may render these actions fruitless anyway. The order in which to process the stores becomes more complex when general assumptions do not hold, such as the persistence of data in memory between reboots, or the persistence of data in memory for extended periods of time (Chow. 2005).

A secondary problem with interacting with the machine at incident response time is a trust issue in using commands from a suspect machine. Techniques commonly used in malware like rootkits and spyware (binary byte patching, application replacement, system call hooking, etc) can alter the output of commands and applications in ways that make the detection of the alteration difficult. Commands may be issued from a more trusted media, such as a CDROM, but in some cases even these perceived to be trusted, read-

only binaries can be subverted¹⁴. Due to scalability issues, this type of subversion technique is likely to only take the most popular response tools into account.

Related to the act of adding a process to the process table, and the argument of cost versus the benefit of doing so, is the concept of information longevity. For example in most file systems when a file is deleted¹⁵, typically the allocation units for the file are marked as available for use and the contents of these units are not cleared. While the block may appear as empty or inaccessible from a file system or operating system point of view, the data has not been cleared and can be accessed by alternate means. This is a primary method of data recovery for a variety of forensic tools. Similar in concept to the example of the extended longevity of a file, all objects have lifetimes. Memory contents may change more often given an active user or active processes, but this depends on the amount of memory, the existence of some sort of paging file and a number of other factors.

Even if certain procedures are followed in the incident response process before the power is removed, some valuable information might not be obtained. Two primary examples are

¹⁴ Some particular types of rootkits, such as hacker defender, actually attempt to sense popular rootkit detection techniques in order to avoid detection. "To overcome some of the countermeasures implemented by Holy Father and other rootkit authors, the latest version creates a randomly named copy of itself that runs as a Windows service. This approach is effective, but Russinovich and Cogswell acknowledge, "It is theoretically possible for a rootkit to hide from Rootkit Revealer. However, this would require a level of sophistication not seen in rootkits to date" (Dillard, 2005)

¹⁵ Deleted from the filesystem, not from the Operating System. Deleted in this context, typically means that the OS is not capable of recovering the information. When using a recycle bin model, a file is not deleted until it has been removed from the recycle bin.

purely memory resident malware¹⁶ and encryption keys stored in memory. Some tools like The Metasploit Project may use techniques like direct memory injection to load OS modules without leaving any evidence on the disk.

The main purpose of analyzing volatile data stores is to reduce the impact the investigator subjects upon potential evidence while increasing the amount and credibility of the evidence that is acquired. Even so, it is important to point out that in some instances even the acquisition process can be subverted. Situations such as the examination of a system that has a rootkit present create further challenges for the investigator. Even in such a situation, the availability of RAM contents at the time of power removal (and a forensic duplicate of the physical drive) would likely be beneficial to an investigator, even if the RAM was incomplete. A greater difficulty lies in the potential ability of an untrustable system to wholly deny access to data stores.

Typical incident response consists of running a series of commands – each starting its own process, the output of which is stored on a secondary device so as to not potentially overwrite data on the disk. Each command will result in creating at least one new process which may overwrite latent data in RAM much like creating a new file may overwrite latent data on disk. A contemporary listing of typical incident response steps can be found in Table 1: Common Incident Response Steps.

¹⁶ Worms, Virii, Trojans, rootkits, spyware and the like. Examples would be Nimda or SQL slammer.

| Table 1: Common Incident Response Steps (Nolan, O’Sullivan, Branson, Waits) | | |
|---|--|--|
| | Windows | Linux |
| System Profile | systeminfo.exe, psinfo | /proc (version, uptime, meminfo, filesystems, cpuinfo), uname |
| Date and Time | netstat, date, time | netstat, date |
| uptime | psuptime, net statistics | uptime, w |
| Runing Processes | netstat, pulist, tlist, pslist, listdlls | ps, w, top, fuser, modules.conf, ldd, ls |
| Open Files, startup, clipboard | dir, afind, macmatch, autoruns, handle, pclip | ls, find, lsof, file, /etc/rc* directories, chkconfig, inittab, cron, at |
| Users | net users, psloggedon, ntlast, dumpusers | who, last, lastlog, /etc/passwd, /etc/shadow |
| Network information | ipconfig, fport, psservice, promiscdetect, netstat, nbstat, net, arp | ifconfig, netstat, /var/log/messages, arp |

There are still other areas that blur the line between the analysis of a static store and a volatile store. Swap space, for example, resides on disk either in an allocated form (such as a pagefile in Windows) or somewhat more raw form (such as a swap partition in Linux). Upon removal of power from a system, swap space may still exist largely intact¹⁷. However, the analysis of such space will be similar to the raw analysis techniques presented here and less similar to traditional techniques related to forensic procedures applied to a typical file system. Traditional techniques may allow the detection or recovery of the pagefile from the file system, but the interpretation of the contents will be much more analogous to RAM analysis than file analysis. Furthermore, the availability of both swap information as well as an image of the RAM will allow

¹⁷ Largely is a subjective term and no assumptions are made as to the exact or expected percentage of intact swap information. Each system will have different results. In fact each system will even show different results on subsequent experiments. The data that will remain past the removal of power will depend greatly on the state and configuration of the system.

some comparison. Obviously the level of comparison that can be made will depend entirely on the system in question due to the aforementioned state of swap information.

Memory Acquisition

Generally speaking, data stored in a volatile data store is, as the name implies, volatile in nature. Introductory level students of computer science are taught early in their education that the difference between a hard disk technology and RAM technology is that the data in RAM is lost when power is removed from the system. This theory also falls in line with current movements in the Law Enforcement sector, and is fundamental to the need presented in this thesis for adaptation of current incident response processes. However, it has recently been proven that at least some hardware retains data in RAM for certain periods of time. For example, an IBM T30 Thinkpad laptop may retain RAM contents for as long as 30 seconds without power (Chow, Pfaff, Garfinkel, Rosenblum. 2005). Similarly, samples taken as part of this research also clearly show that RAM data survives reboots.

Without a specialized hardware tool designed to rapidly copy data from the RAM immediately after power down, or designed to clamp on to the memory stores of a running machine and duplicate in stream¹⁸, the examiner must resort to using provided methods to access volatile stores. These methods may be provided at different levels of abstraction and likewise offer different granularities and insight into the data present.

¹⁸ Such as the PCI described in A Hardware Based Memory Acquisition Procedure for Digital Investigations (Carrier, Grand).

For recovery and testing purposes, many versions of Windows can be configured to perform a memory dump upon system crash, called a crash dump. This functionality is typically dictated graphically by choosing the “Startup and Recovery” button under the “Advanced Tab” of “My Computer” properties page. The “Write debugging information” field can be set to Complete, Kernel, or Small style memory dumps. These settings can also be manipulated using the Windows registry:

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\CrashControl
```

```
CrashDumpEnabled REG_DWORD 0x0 = None
CrashDumpEnabled REG_DWORD 0x1 = Complete memory dump
CrashDumpEnabled REG_DWORD 0x2 = Kernel memory dump
CrashDumpEnabled REG_DWORD 0x3 = Small memory dump (64KB)
```

Related keys of interest are:

```
AutoReboot REG_DWORD 0x1
DumpFile REG_EXPAND_SZ %SystemRoot%\Memory.dmp
MinidumpDir REG_EXPAND_SZ %SystemRoot%\Minidump
```

Complete memory dumps include the entire contents of physical memory and are by default eventually saved to %SystemRoot%\Memory.dmp (see page 35 for more details). Complete memory dumps will require a swap file larger than the physical RAM size of the machine plus 1 MB (to allow the addition of a file header and some kernel variable values)¹⁹. In situations involving ‘large’ amount of RAM, special considerations must be taken. There are several workarounds requiring registry, and boot

¹⁹ An entire megabyte is not required, but the smallest increment of the Windows paging system is 1 MB.

modifications to allow complete memory dumps for Windows 2000 based systems with more than 2 GB of RAM. Complete memory dumps are not possible with Windows 2000 based systems using Physical Address Extension or with more than 4 GB of RAM (due to the page file size limitation of 4095 MB and the complete memory dump size for 4 GB of RAM to be 4096 + 1 MB). Many administrators might readily point out that it is possible to have more than 4095 MB of paging space. While this is true, it is achieved through multiple page files and 4095 MB is the maximum size for each individual file.

| Table 2: : Windows pagefile sizing by Architecture | | | |
|---|-------|--------|--------|
| | x86 | x64 | IA-64 |
| Maximum size of a paging file | 4 GB | 16 TB | 32 TB |
| Maximum number of paging files | 16 | 16 | 16 |
| Total paging file size | 64 GB | 256 TB | 512 TB |

| Table 3: Maximum RAM support by OS²⁰ | |
|--|------------------|
| OS Version | Maximum RAM (GB) |
| Windows NT | 4 |
| Windows 2000 Professional | 4 |
| Windows 2000 Standard Server | 4 |
| Windows 2000 Advanced Server | 8 |
| Windows 2000 Datacenter Server | 32 |
| Windows XP Professional | 4 |
| Windows Server 2003 Web Edition | 2 |
| Windows Server 2003 Standard Edition | 4 |
| Windows Server 2003 Enterprise Edition | 32 |
| Windows Server 2003 Datacenter Edition | 64 |

A kernel memory dump only dumps memory pertaining to kernel level processes. Kernel memory dumps require the primary volume's pagefile to be at least approximately 1/3 of

²⁰ All Windows limits pertain to x86 32 bit architecture not observing /3G, /AWE or /USERAV boot switches. Some such as 64 bit, Server 2003 with Service Pack 1 can support up to 1024 GB of RAM.

the system's physical RAM. This type of dump differs from a complete dump in that only kernel level processes (the OS kernel, device drivers, and system level programs, but NOT user programs or unallocated memory) and is thus much faster and space efficient, but contains less information. The %SystemRoot%\Memory.dmp location is also used for kernel memory dumps. Each time a STOP error occurs, the Memory.dmp file is replaced with a new version pertaining to the most recent crash. This replacement and thus loss of prior information affects both complete and kernel memory dump types.

Small memory dumps (aka minidumps) are, as the name implies, much smaller – 64 KB in 32-bit systems. Minidumps only require 2 MB of page file space, and instead of including full contents of RAM or kernel allocated RAM, minidumps include at a minimum the STOP message, a list of drivers, processor context, process and thread context, and a kernel mode call stack. Minidumps are saved as individual files in the %SystemRoot%\minidump directory and are named according the date on which the error occurred (for example, Mini070506-01.dmp for the first minidump on July 05, 2006).

The facts that subsequent minidumps do not overwrite previous minidumps, and that minidumps are relatively small make this type of dump desirable to administrators. However, the lack of RAM data makes the minidump type of memory dump less desirable for the analysis techniques described in this thesis (though the existence of information in the minidumps may still hold valuable forensic information). If minidump

style dumps are the only type of dump available, some simple information may be discerned. Access to core binaries (such as `ntoskernel.exe`) from the suspect system and possibly access to symbols²¹ for the examined version of Windows may allow additional interpretation of the information in a minidump, but it will pale in comparison to the information attainable from a larger memory dump.

Client platform Windows operating systems default to minidump style, and server platform Windows operating systems default to complete style dumps. On XP and 2003 Server platforms minidumps are created in addition to the complete or kernel dump. If a complete or kernel dump file is available, a minidump can be created from that complete or kernel dump using `.dump /m` in WinDbg.

All memory dumps, regardless of source (crash induced or somehow instantiated) require space for storage. In the most raw form, a total image of RAM would require at least the same space as the physical RAM (1 MB if in the Microsoft DMP format). Since this data is eventually written to disk, the possibility exists that less volatile evidence stored on disk will be over-written. If possible, the RAM could be saved to a non-suspect device (e.g. a tape drive). If not possible, the perceived benefit of saving system RAM must be weighed against the possibility of overwriting potential evidence.

²¹Symbols are basically a way to give more information to a debugging tool, like function and variable names, that would not normally be available in a compiled version of software. Further information on symbols appears later.

When a memory dump is forced by a crash, the Windows kernel actually loads a miniport driver to write the dump contents directly to sectors that are occupied by the pagefile.

Because of this, contents of the pagefile will contain a DMP formatted file structure that can be later extracted using forensically sound procedures if power is removed from the system after the dump has completed but before the operating system has been reloaded.

The `Memory.dmp` file in the `%SystemRoot%` directory is not created until after the system reboots²². After the operating system has started to reload, the Session Manager Subsystem (SMSS) user process enables paging upon boot. If SMSS determines a DMP memory dump is found in the pagefile, SMSS instructs the kernel to mark the parts of the pagefile occupied by a DMP as unusable. Later in the boot process, WinLogon checks for DMP memory dump data in the pagefile. If found WinLogon spawns Savedump to extract the file and store it the `%SystemRoot%`. In situations involving large quantities of RAM this detection and copy can significantly slow down the boot process following a crash dump. Because of this, when using the Windows crash dump method of obtaining a memory dump, a RAM to pagefile comparison will not be possible. The automatic creation of a minidump from the complete memory dump happens after the system has finished rebooting. It is important to note that even though the `MEMORY.DMP` file is not created until after the system reboots, the information is actually written to disk. If a

²² This is very easy to observe. Simply follow the MS instructions, as outlined in this thesis to setup a test machine for a “Full Memory Dump” then use one of the techniques presented to initiate a crash dump. Instead of allowing the machine to reboot to the OS, reboot using a LiveCD. Browsing the file system will show that `MEMORY.DMP` does not yet exist, even though the physical memory was written to disk (as evidenced by the blue screen after the crash). Upon rebooting into the OS the `MEMORY.DMP` file will be created.

system crash occurs (or is forced), it is preferred to not let the system reboot which would cause multiple changes to occur to the suspect non-volatile stores. Not only would this reboot alter expected things like OS timestamps, but also overwrite a large amount of unallocated space with the newly created MEMORY.DMP file.

The DMP format is a proprietary Microsoft file type, but the structure of the additional information present in a DMP file is easily discerned through the use of tools provided by Microsoft. The DMP header is commonly stated to have 1MB for header information

Figure 3: DMP File Header

| Offset | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F | |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------------------|
| 00000000 | 50 | 41 | 47 | 45 | 44 | 55 | 4D | 50 | 0F | 00 | 00 | 00 | 93 | 08 | 00 | 00 | PAGEDUMP...." |
| 00000010 | 00 | 00 | 03 | 00 | 00 | 20 | 0D | 82 | F0 | D9 | 46 | 80 | B0 | DC | 46 | 80 |,øÛ€°Û€ |
| 00000020 | 4C | 01 | 00 | 00 | 01 | 00 | 00 | 00 | E2 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | L.....â..... |
| 00000030 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 41 | 47 | 45 |AGE |
| 00000040 | 50 | 41 | 47 | 45 | 50 | 41 | 47 | 45 | 50 | 41 | 47 | 45 | 50 | 41 | 47 | 45 | PAGEPAGEPAGEPAGE |
| 00000050 | 50 | 41 | 47 | 45 | 50 | 41 | 47 | 45 | 50 | 41 | 47 | 45 | 00 | 41 | 47 | 45 | PAGEPAGEPAGE.AGE |
| 00000060 | 58 | F3 | 46 | 80 | 04 | 00 | 00 | 00 | 8C | FE | 01 | 00 | 01 | 00 | 00 | 00 | XóF€.....Ep..... |
| 00000070 | 1F | 00 | 00 | 00 | 21 | 00 | 00 | 00 | 7E | 00 | 00 | 00 | 00 | 01 | 00 | 00 |!....~..... |
| 00000080 | FF | 0E | 00 | 00 | 00 | 10 | 00 | 00 | F0 | EE | 01 | 00 | 50 | 41 | 47 | 45 | ÿ.....Œí..PAGE |
| 00000090 | 50 | 41 | 47 | 45 | 50 | 41 | 47 | 45 | 50 | 41 | 47 | 45 | 50 | 41 | 47 | 45 | PAGEPAGEPAGEPAGE |
| 000000A0 | 50 | 41 | 47 | 45 | 50 | 41 | 47 | 45 | 50 | 41 | 47 | 45 | 50 | 41 | 47 | 45 | PAGEPAGEPAGEPAGE |
| 000000B0 | 50 | 41 | 47 | 45 | 50 | 41 | 47 | 45 | 50 | 41 | 47 | 45 | 50 | 41 | 47 | 45 | PAGEPAGEPAGEPAGE |
| 00000F70 | 50 | 41 | 47 | 45 | 50 | 41 | 47 | 45 | 50 | 41 | 47 | 45 | 50 | 41 | 47 | 45 | PAGEPAGEPAGEPAGE |
| 00000F80 | 50 | 41 | 47 | 45 | 50 | 41 | 47 | 45 | 01 | 00 | 00 | 00 | 50 | 41 | 47 | 45 | PAGEPAGE....PAGE |
| 00000F90 | 50 | 41 | 47 | 45 | 50 | 41 | 47 | 45 | 50 | 41 | 47 | 45 | 50 | 41 | 47 | 45 | PAGEPAGEPAGEPAGE |

(Russovich, 2005)(KB 309773), but for the purpose of this thesis only the first 4KB is useful. From this first 4 KB extra metadata can be discerned (e.g. Version and Build of the Windows OS that created the dump, the number of CPUs, the system architecture, and the type of dump created (kernel or complete)). Some of this information is in addition to what can be detected from the memory image, while other parts of this information can be used to validate the results. At the very least the presence of the header allows the program itself to adjust the detection processes depending on input. The indicated portions of Figure 3 show the file signature (0x5041474544554D50 – “PAGEDUMP”), the “Major” OS version (0xF000000 = 15), the “Minor” OS Build number (0x93080000 = 2195 – Windows 2000), the architecture (0x0000014C = 338 – i386), the number of processors (0x00000001 = 1) and the type of dmp file (0x00000001 = 1 – Complete [2 would be kernel]).

Tools distributed from Microsoft for debugging Windows-based software, such as the I386kd tool²³, the Userdump OEM tool, or similarly the crashdump utility built into the kernel, create one of the proprietary “dmp” files designed to help discern the cause of bugs and crashes. While these tools are not designed with forensics in mind, they can still provide information from RAM. Most of the tools designed with debugging in mind will write the volatile information to a non-volatile store, which is forensically poor since the non-volatile store is usually also suspect, and this action could potentially overwrite non-volatile evidence. A middle ground solution may be to introduce another storage

²³ Found on the Windows 2000 Support CD-ROM

location to accept the dump (e.g. a network share or portable storage device provided by the investigator). Different techniques must be weighed against existing policy, and may well depend on the circumstances of the particular incident. Mapping drive letters to network shares and introducing external devices (like USB hard drives) both have impact on the system. The best case might be that the new hardware or shares will be enumerated at various places on disk and thus the procedure must be documented by the responder to avoid later misinterpretation by an analyst. The worst case might be accidentally causing executables to be run (possibly via autorun) or even system failure.

Most kernel debugging tools, both Microsoft distributed and third party, require installation and configuration. Some even require special connections and/or hardware. Both installation of software and special connections to hardware may have unacceptable, adverse impacts on the state of the system. For this reason kernel debugging solutions are often not a viable choice for most investigations. Some tools or portions of packages are in a portable executable form and can be executed independent of the debugging installation. Such tools might become a valuable part of a toolkit²⁴ designed for the initial incident response (first responder). Other tools such as dumpchk.exe which verifies the correct creation of the memory dump might appear to be useful, but dumpchk.exe only works on minidump files and provides no insight in the inspection of complete or kernel memory dumps.

²⁴ Different packages and toolkits have varying license agreements. It may not be within the rights of the purchaser to use these tools in this manner. Research done for the purpose of this thesis was for proof of concept and education purposes only. Users must verify similar usage against respective license agreements.

Since Windows source code is not available to the typical developer, code level analysis is not available when troubleshooting even the simplest kinds of errors. To aid the debugging efforts of developers, Microsoft distributes what is known as “symbols” for multiple versions of Windows. Symbols allow a debugger to correlate application execution statements with function names, line numbers, etc. without the need for source code. At compile time (actually linking) the compiler can create symbol information along with the compiled executable. “Symbol files hold a variety of data which are not actually needed when running the binaries, but which could be very useful in the debugging process. Typically, symbol files might contain: Global variables, Local variables, Function names and the addresses of their entry points FPO data, and Source-line numbers”(Microsoft Corp. 2006). The exclusion of this information in the compiled binary is purposeful in order to create smaller, faster binaries. While possession of symbols is not required to debug or reverse engineer a compiled product, it can greatly reduce the complexity.

Obtaining live dumps of RAM can be problematic. In addition to altering the contents of memory (no matter how slightly) by performing the dump, the memory will also be changing as the system continues to run due to other running processes. This, paired with the fact that it will take some time to actually obtain the dump, means that the resulting dump will not actually reflect the state of RAM an exact point of time, but rather a time sliding view of memory. Faster imaging will result in a smaller time window, and on all but the most active systems RAM is unlikely to change substantially during the imaging process, but the fact remains that it will change – it is just a question of how much.

Actions similar to some of the above kernel debugging techniques can be used on a live system. Windbg's .dump can be used in conjunction with Sysinternals' Livekd in order to create a dump of a running kernel²⁵ (Russovich, Solomon. 2005). Similar results should be possible using more robust tools such as SoftICE or IDAPro.

George Garner's dd modifications allow for the /Device/PhysicalMemory object to be copied using the dd tool (used in many Open Source forensic packages, such as the Helix LiveCD). Invoking the dd command is not much different in a Win32 environment than in a Linux environment. The only part that might appear alien to one who has used dd before is the input, which is an access method to the device "PhysicalMemory."²⁶

```
dd if=\\.\Device\PhysicalMemory of=memory.bin bs=4096
```

Using dd has many advantages, there is no need to install software, it has a small executable size for lower impact and easy portability, a simple file structure, and is open source. Because the file is stored in a "RAW" format, low level analysis is simple – file offsets are the same as memory offsets, however this can also be viewed as a negative

²⁵ Live debugging is built into the windows debugger, but requires a secondary debug system to be attached and the debug target must be booted with the /DEBUG switch. "Local" kernel debugging is possible with XP / 2003 Server but does not allow the .dump and thus does not pertain to the debugging needs presented here.

²⁶ Notice the use of the bs option for blocksize. This option should be set to 4096, the size of a page.

The actual command used for the results presented in this text was:

```
dd if=\\.\Device\PhysicalMemory of=e:\memory\OSTYPE\OSTYPE.dd bs=4096
conv=noerror --md5sum --verifymd5 md5file=e:\memory\OSTYPE\OSTYPE.md5 --
log=e:\memory\OSTYPE\audit.log
```

where OSTYPE was user supplied at the time and varied by OS used. E: mapped to an external USB mass storage device.

because this does not preserve the DMP file format in which Windows natively dumps, and thus cannot be used with tools expecting DMP formatted files. This method also has the advantage of operating similarly to a dd memory dump performed on a Linux system, which gives the tool familiarity for the user across platforms. Unfortunately it appears that this technique will not be particularly useful in the future. Microsoft has decided to change the functionality of the `\Device\PhysicalMemory` object in Server 2003 SP1. Usermode access to `\Device\PhysicalMemory`, which is required for user level program access such as the dd tool, is not permitted. Kernelmode access is still granted, but this gives little peace of mind to the forensic investigator as kernelmode access would require either installation of an imaging program or pre-meditated configuration, and neither case is likely in most situations.

For the sake of completeness and for testing purposes it should be pointed out that memory dumps can also be forced. For the purpose of incident response, many of these methods are not preferred because they typically either require pre-meditation or have an impact to a non-volatile store that is unacceptable. These methods should still be of interest to the incident responder, if for no other reason than that of tool validation. Forcing memory dumps on known systems allows for the testing of tools that will be used in the field, and repetitive, scheduled testing of software and hardware to validate manufacturer claims should be an integrated part of operations. The aforementioned kernel debugging tools can often be used to force a memory dump. Similarly, freely distributed tools such as UserDump can dump the memory space of a single process, but

this does not directly apply to the objectives set forth here and is only mentioned for the sake of completeness.

For Windows 2000 based operating systems (2000, XP Pro, Server 2003 and variants) there is actually a built in way to force a system crash, and thus a memory dump (KB 244139). A REG_DWORD named `CrashOnCtrlScroll` with a value of 1 must be added (or edited) to:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\i8042prt\Parameters
```

This registry update requires a system restart. The keyboard driver will now have added functionality. When the right Control key is held and Scroll Lock is pressed twice²⁷, the crashdump utility will be executed and the infamous Windows Blue Screen will be shown with a defined STOP code of 0x000000E2, end-user generated crashdump. This of course requires that the system was configured to perform one of the three previously mentioned types of crash dumps upon a system crash.

A fairly low-impact method developed by Mark Russinovich involves loading a small driver (~7 kb) called `MyFault.sys` and a user level application `NotMyFault.exe` (~50 kb). The user executable issues calls to the kernel loaded driver which crashes the system on behalf of the user level executable in various ways. This pair of files can force a number of crash conditions in Windows and thus can force the creation of a memory

²⁷ Due to the implementation of this method in the `i8042prt.sys` driver, this method likely does not work on USB keyboard. Similarly, if a virtual machine is used and the appropriate client tools are installed on the virtual machine it is quite likely that the `i8042prt.sys` driver is replaced with a virtual machine variant that does not exhibit this functionality. Also, many KVMs intercept `scrlck` pressed twice to provide a user menu which may prevent the keycodes from actually reaching the system.

dump if the machine is appropriately configured. The tool is interactive and will alter the state of the suspect machine, but in the event that live imaging of RAM is not possible (as seems to be the case with 2003 Server SP1) this may be a lesser impacting method of obtaining system state information than interactively gathering information from the running suspect machine.

Some memory dumps (crash dumps) can actually be interrupted by a BIOS level system restart. For example, for some time some Compaq Server systems have had a high availability feature that detects when a system stops responding and forces a system reboot (AlphaServer Comparison Chart, December 1996). In such a situation where the machine essentially causes a soft restart autonomously, the creation of a full memory dump after a system crash will likely be interrupted. A likely course of action would be to disable this feature in BIOS when the system restarts and use a very minimal LiveCD to acquire memory in hope that some of the memory contents have persisted through the reboot cycle. Unfortunately it is likely impossible to detect whether this feature is enabled on a running machine prior to forcing the memory dump.

Memory Analysis

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 volatile data. OS type and patch level are among the foremost important factors.

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 strings²⁸ analysis may be used by an investigator to simply glance at a subset of data looking for something that might provide some direction. Experiments run in conjunction with this project showed that, on average, a cleanly booted workstation with 512 MB of RAM would produce 50-80 MB of 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 means 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 worse if only a hex view analysis is performed without the aid of the strings tool.

Original Analysis Goals:

- Must work on dd-style dumps (preferred, though not hopeful for the future due to 2003 sp1) and on Microsoft DMP Complete style dumps.
- Must be simple to use, since the target user base will typically be law enforcement, not computer scientists.
- 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)

²⁸ Strings is a program developed for UNIX and ported to Windows that allows the extraction of “printable sequences of characters” from a file – no matter what type of file. It is commonly used on binary files to aid in deductions to be made about the binary. (Strings Man Page)

- Must work on multiple versions of Windows and Linux.

The Windows EPROCESS structure and significance

As previously stated, 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, the EPROCESS structure can be enumerated using a kernel debugger. (e.g. using the Windows debugger to enumerate fields by issuing a `!processfields, dt _eprocess` or `dt nt!_eprocess` command.)

Substructures can also be enumerated in this way. From the information gleaned from the debugger (and available in Appendix B, C and D) a model for the EPROCESS and subsequent structures can be created as seen in Table 4: EPROCESS Structure. 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 (See Appendix B: EPROCESS dumps of a live typical system – XP SP2).

Table 4: EPROCESS Structure
(Ruslinovich, Solomon. 2005)

| EPROCESS Element | Purpose |
|--|---|
| Kernel process (KPROCESS or PCB) block | Common dispatcher object header, pointer to the process page directory, list of kernel thread (KTHREAD) blocks belonging to the process, default base priority, quantum, affinity mask, and total kernel and user time for the threads in the process. |
| Process identification | Unique process ID, creating process ID, name of image being run, window station process is running on. |
| Quota block | Limits on nonpaged pool, paged pool, and page file usage plus current and peak process nonpaged and paged pool usage. (<i>Note: Several processes can share this structure: all the system processes point to the single systemwide default quota block; all the processes in the interactive session share a single quota block Winlogon sets up.</i>) |
| Virtual address space descriptors (VADs) | Series of data structures that describe the status of the portions of the address space that exist in the process. |
| Working set information | Pointer to working set list (MMWSL structure); current, peak, minimum, and maximum working set size; last trim time; page fault count; memory priority; outswap flags; page fault history. |
| Virtual memory information | Current and peak virtual size, page file usage, hardware page table entry for process page directory. |
| Exception local procedure call (LPC) port | Interprocess communication channel to which the process manager sends a message when one of the process's threads causes an exception. |
| Debugging LPC port | Interprocess communication channel to which the process manager sends a message when one of the process's threads causes a debug event. |
| Access token (ACCESS_TOKEN) | Executive object describing the security profile of this process. |
| Handle table | Address of per-process handle table. |
| Device map | Address of object directory to resolve device name references in (supports multiple users). |
| Process environment block (PEB) | Image information (base address, version numbers, module list), process heap information, and thread-local storage utilization. (<i>Note: The pointers to the process heaps start at the first byte after the PEB.</i>) |
| Win32 subsystem process block (W32PROCESS) | Process details needed by the kernel-mode component of the Win32 subsystem. |

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

One part of the EPROCESS structure that is easy to identify with is the timestamp information²⁹. 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, 1601³⁰. Once the location of a FILETIME is known, some conversion must take place to make this a usable timestamp for investigative purposes. A benefit of decoding a timestamp allows for manual comparison to disk times to process times for rough estimating and correlation.

²⁹ While one might assume that this should also hold true for Threads that have similar timestamp offsets, in practice this appears to not be the case.

³⁰ The FileTime structure format of 100 nanosecond intervals (aka a “tick”) might seem counter-intuitive, but this allows a range of more than 30,000 years to be represented in 64 bits at a finer than one second resolution. 1000 nanosecond intervals would only yield about 200 years, 1 nanosecond intervals would likely be too small a time interval for some processor clock speeds. Standard *nix only allows one second resolution since 1/1/1970. The 1970 date for the “birth of Unix” makes some sense, but I am not sure about the significance of 1601. Converting from the Microsoft FileTime 100 nanosecond structure to the *nix 1 second structure does lose some information, but converting the other direction does not add any information and for the purposes of this text, one second resolution is sufficient.

Below are two offsets from a Windows XP SP2 EPROCESS structure (from Appendix B: EPROCESS dumps of a live typical system – XP SP2). 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
```

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.) (Russovich, 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. The procedure is described in the Virtual Addressing and Paging section and an example can be found in Appendix E: Decoding a Process Owner.

Finding processes in memory image

When searching through a memory dump, the most complete way to search for process structures will be to start assuming each byte is the first of a process structure until further offsets show that that was indeed not a first byte of a process, then to shift one byte and repeat the process. When considering the aforementioned sizes of RAM available in today's OSs, this may very well become computationally impractical. On the other hand it is not safe to assume that all process structures will be allocated on a page boundary, but it might be safe to assume certain other boundaries (such as an 8 byte boundary for Windows³¹). 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.

³¹ "If the driver requests fewer than PAGE_SIZE bytes, ExAllocatePoolWithTag allocates the number of bytes requested. If the driver requests PAGE_SIZE or greater bytes, ExAllocatePoolWithTag allocates a page-aligned buffer that is an integral multiple of PAGE_SIZE bytes. Memory allocations of less than PAGE_SIZE do not cross page boundaries and are not necessarily page-aligned; instead, they are aligned on an 8-byte boundary" (Six Tips for Efficient Memory Usage)

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 PDI, or kernel memory address which must be mapped above 0x80000000).

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

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. (See Appendix F: Offset Deltas by Service Pack for more offset examples)

Windows XP SP2 EPROCESS field (from Appendix B):

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

Windows 2000 SP4 EPROCESS field (from Appendix C):

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

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 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 sturture location in RAM). Finally, the complete memory dump can actually be processed identically to that of a dd style dump because the DMP header size is a multiple of the page size (or more to the point, the first location of to the contents of the RAM dump falls on the 8 byte scan boundary.)

Conclusions

Acquiring a RAM image from a suspect system is potentially different for every circumstance. The more information presented to the party performing the acquisition the better. Some cursory inspection can potentially reveal some items like general operating system type, and physical connections to the system.

On a pre-Server 2003 SP1 Windows system, the preferred way of acquiring a RAM image is to use a imaging tool from a trusted source to copy RAM through the PhysicalMemory object. The image may be stored to a local device connected to the system (such as Firewire or USB mass storage), or through the network interface.

A secondary method would be to force a system crash and thus a crash dump of physical memory to disk. This method will not allow a comparison to the pagefile, but will allow the contents of physical memory to be acquired and analyzed later. A simple registry modification paired with a small program that can force a crash could easily be placed on read only media such as CDROM, floppy disk, or some types of USB memory sticks. In this case mass storage would not be required as the memory image would be stored to the suspect system disk. The actual acquisition of the image would happen when the physical disk is imaged later in a typical forensic process.

The proof of concept PERL script shows that information about the state of a system can be found postmortem. At the very least, Task Manger functionality can be simulated from a RAM image, and in some cases more information is available than Task Manger is capable of reporting. 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.

Direction

The PERL script could be improved in order to make it more likely to be used by mainstream responders. The current state of the tool is definitely proof of concept and should not be considered production level. Desirable features may include automatic detection of the OS from which the RAM was acquired, detection of popular dump

formats (DMP) versus raw RAM capture, or the extraction of selected processes memory space. Automated OS detection can be done a variety of ways, trying all possible offsets looking for number of processes detected, then comparing with known required or default processes for different OSes.

The proof of concept only emulates Task Manager information. The creation of similar tools to obtain other popular incident response information (like current network information, open files, etc) should be explored.

Several interesting directions present themselves in relation to the work done here.

Virtual machines, particularly VMWare™, allow for essentially instant RAM acquisition because a virtual machine has its “Virtual RAM”³² stored as a file on the host operating system. If a virtual machine is suspended, the entire content of RAM for that virtual machine exist as a logical file that can be copied. Other directions may exist with other virtual machine related topics like virtual machine monitors, Parallels™, the Trusted Computing Platform Alliance (TCPA) / Trusted Computing Group (TCG) / Palladium, and hypervisor architecture.

³² The “Virtual RAM” is represented as physical RAM to the Virtual Machine. This is much different that Virtual Memory discussed elsewhere in this text.

Appendix A: Methodology notes

Samples were taken from different OS installations on the same hardware utilizing the Nebraska University Consortium on Information Assurance's (NUCIA) Security Technology Education and Analysis Laboratory (STEAL). These samples were taken from a set of IBM Intellistation MPro model 6220 systems, with 512 MB of RAM.

For each RAM image the machine was left without power for more than 15 minutes then powered on and the RAM was dumped using one of the techniques outlined in this paper.

Each Windows RAM image was created according to the following process:

1. Use Symantec Ghost to restore a known good installation of the OS.
2. Edit the registry to include the CrashOnCtrlScroll and Complete memory dump keys
3. Shutdown the system
4. Leave without power for 15 minutes.
5. Power on and log in.
6. Attach external USB mass storage device (hold down left shift to prevent Autoplay)
7. Insert Helix 1.7 CDROM (allow Autoplay)
8. Use the dd utility found on Helix to perform a RAM dump to the USB device.
9. Unmount CDROM and USB device.
10. Force a crash using the Ctrl-ScrLck-ScrLck method
11. Reboot
12. Re-attach USB device and copy the Memory.dmp, ntoskernel and any minidump.dmp files.

To study RAM persistence, the same machine was used but the machine was restarted with the power removed for various durations between shutdown and startup.

Additionally, samples were taken from random machines including: IBM Thinkpad R52, Dell Inspiron 8600, and Gateway 2000 E-4200, for non-baseline tests, and generally for the availability of a more diverse set of data to inspect.

Appendix B: EPROCESS dumps of a live typical system – XP SP2

Appendices B, C and D show kernel debugger output of data structures found in the nt kernel. The EPROCESS and ETHREAD are two structures that are focused on in this text for the identification of processes in memory. To aide the reader, portions of these structures are shown in **bold**.

```
kd> dt _eprocess
ntdll!_EPROCESS
+0x000 Pcb : _KPROCESS
+0x06c ProcessLock : _EX_PUSH_LOCK
+0x070 CreateTime : _LARGE_INTEGER
+0x078 ExitTime : _LARGE_INTEGER
+0x080 RundownProtect : _EX_RUNDOWN_REF
+0x084 UniqueProcessId : Ptr32 Void
+0x088 ActiveProcessLinks : _LIST_ENTRY
+0x090 QuotaUsage : [3] Uint4B
+0x09c QuotaPeak : [3] Uint4B
+0x0a8 CommitCharge : Uint4B
+0x0ac PeakVirtualSize : Uint4B
+0x0b0 VirtualSize : Uint4B
+0x0b4 SessionProcessLinks : _LIST_ENTRY
+0x0bc DebugPort : Ptr32 Void
+0x0c0 ExceptionPort : Ptr32 Void
+0x0c4 ObjectTable : Ptr32 _HANDLE_TABLE
+0x0c8 Token : _EX_FAST_REF
+0x0cc WorkingSetLock : _FAST_MUTEX
+0x0ec WorkingSetPage : Uint4B
+0x0f0 AddressCreationLock : _FAST_MUTEX
+0x110 HyperSpaceLock : Uint4B
+0x114 ForkInProgress : Ptr32 _ETHREAD
+0x118 HardwareTrigger : Uint4B
+0x11c VadRoot : Ptr32 Void
+0x120 VadHint : Ptr32 Void
+0x124 CloneRoot : Ptr32 Void
+0x128 NumberOfPrivatePages : Uint4B
+0x12c NumberOfLockedPages : Uint4B
+0x130 Win32Process : Ptr32 Void
+0x134 Job : Ptr32 _EJOB
+0x138 SectionObject : Ptr32 Void
+0x13c SectionBaseAddress : Ptr32 Void
+0x140 QuotaBlock : Ptr32 _EPROCESS_QUOTA_BLOCK
+0x144 WorkingSetWatch : Ptr32 _PAGEFAULT_HISTORY
+0x148 Win32WindowStation : Ptr32 Void
+0x14c InheritedFromUniqueProcessId : Ptr32 Void
+0x150 LdtInformation : Ptr32 Void
+0x154 VadFreeHint : Ptr32 Void
+0x158 VdmObjects : Ptr32 Void
+0x15c DeviceMap : Ptr32 Void
+0x160 PhysicalVadList : _LIST_ENTRY
+0x168 PageDirectoryPte : _HARDWARE_PTE_X86
+0x168 Filler : Uint8B
+0x170 Session : Ptr32 Void
+0x174 ImageFileName : [16] UChar
+0x184 JobLinks : _LIST_ENTRY
+0x18c LockedPagesList : Ptr32 Void
+0x190 ThreadListHead : _LIST_ENTRY
+0x198 SecurityPort : Ptr32 Void
+0x19c PaeTop : Ptr32 Void
```

```

+0x1a0 ActiveThreads      : Uint4B
+0x1a4 GrantedAccess      : Uint4B
+0x1a8 DefaultHardErrorProcessing : Uint4B
+0x1ac LastThreadExitStatus : Int4B
+0x1b0 Peb                : Ptr32 _PEB
+0x1b4 PrefetchTrace      : _EX_FAST_REF
+0x1b8 ReadOperationCount : _LARGE_INTEGER
+0x1c0 WriteOperationCount : _LARGE_INTEGER
+0x1c8 OtherOperationCount : _LARGE_INTEGER
+0x1d0 ReadTransferCount  : _LARGE_INTEGER
+0x1d8 WriteTransferCount : _LARGE_INTEGER
+0x1e0 OtherTransferCount : _LARGE_INTEGER
+0x1c8 CommitChargeLimit  : Uint4B
+0x1ec CommitChargePeak  : Uint4B
+0x1f0 AweInfo            : Ptr32 Void
+0x1f4 SeAuditProcessCreationInfo : _SE_AUDIT_PROCESS_CREATION_INFO
+0x1f8 Vm                 : _MMSUPPORT
+0x238 LastFaultCount     : Uint4B
+0x23c ModifiedPageCount  : Uint4B
+0x240 NumberOfVads       : Uint4B
+0x244 JobStatus          : Uint4B
+0x248 Flags              : Uint4B
+0x248 CreateReported     : Pos 0, 1 Bit
+0x248 NoDebugInherit     : Pos 1, 1 Bit
+0x248 ProcessExiting     : Pos 2, 1 Bit
+0x248 ProcessDelete      : Pos 3, 1 Bit
+0x248 Wow64SplitPages    : Pos 4, 1 Bit
+0x248 VmDeleted          : Pos 5, 1 Bit
+0x248 OutswapEnabled     : Pos 6, 1 Bit
+0x248 Outswapped         : Pos 7, 1 Bit
+0x248 ForkFailed         : Pos 8, 1 Bit
+0x248 HasPhysicalVad     : Pos 9, 1 Bit
+0x248 AddressSpaceInitialized : Pos 10, 2 Bits
+0x248 SetTimerResolution : Pos 12, 1 Bit
+0x248 BreakOnTermination : Pos 13, 1 Bit
+0x248 SessionCreationUnderway : Pos 14, 1 Bit
+0x248 WriteWatch         : Pos 15, 1 Bit
+0x248 ProcessInSession   : Pos 16, 1 Bit
+0x248 OverrideAddressSpace : Pos 17, 1 Bit
+0x248 HasAddressSpace     : Pos 18, 1 Bit
+0x248 LaunchPrefetched   : Pos 19, 1 Bit
+0x248 InjectInpageErrors : Pos 20, 1 Bit
+0x248 VmTopDown          : Pos 21, 1 Bit
+0x248 Unused3            : Pos 22, 1 Bit
+0x248 Unused4            : Pos 23, 1 Bit
+0x248 VdmAllowed         : Pos 24, 1 Bit
+0x248 Unused             : Pos 25, 5 Bits
+0x248 Unused1            : Pos 30, 1 Bit
+0x248 Unused2            : Pos 31, 1 Bit
+0x24c ExitStatus       : Int4B
+0x250 NextPageColor      : Uint2B
+0x252 SubSystemMinorVersion : UChar
+0x253 SubSystemMajorVersion : UChar
+0x252 SubSystemVersion    : Uint2B
+0x254 PriorityClass       : UChar
+0x255 WorkingSetAcquiredUnsafe : UChar
+0x258 Cookie             : Uint4B

```

```

kd> !processfields
EPROCESS structure offsets: (use 'dt nt!_EPROCESS')

```

```

kd> dt nt!_ePROCESS

```

```

nt!_EPROCESS
+0x000 Pcb                : _KPROCESS
+0x06c ProcessLock        : _EX_PUSH_LOCK
+0x070 CreateTime         : _LARGE_INTEGER
+0x078 ExitTime           : _LARGE_INTEGER

```

```

+0x080 RundownProtect      : _EX_RUNDOWN_REF
+0x084 UniqueProcessId     : Ptr32 Void
+0x088 ActiveProcessLinks  : _LIST_ENTRY
+0x090 QuotaUsage          : [3] UInt4B
+0x09c QuotaPeak           : [3] UInt4B
+0x0a8 CommitCharge        : UInt4B
+0x0ac PeakVirtualSize     : UInt4B
+0x0b0 VirtualSize         : UInt4B
+0x0b4 SessionProcessLinks : _LIST_ENTRY
+0x0bc DebugPort           : Ptr32 Void
+0x0c0 ExceptionPort       : Ptr32 Void
+0x0c4 ObjectTable         : Ptr32 _HANDLE_TABLE
+0x0c8 Token               : _EX_FAST_REF
+0x0cc WorkingSetLock      : _FAST_MUTEX
+0x0ec WorkingSetPage      : UInt4B
+0x0f0 AddressCreationLock : _FAST_MUTEX
+0x110 HyperSpaceLock      : UInt4B
+0x114 ForkInProgress      : Ptr32 _ETHREAD
+0x118 HardwareTrigger     : UInt4B
+0x11c VadRoot             : Ptr32 Void
+0x120 VadHint             : Ptr32 Void
+0x124 CloneRoot           : Ptr32 Void
+0x128 NumberOfPrivatePages : UInt4B
+0x12c NumberOfLockedPages : UInt4B
+0x130 Win32Process        : Ptr32 Void
+0x134 Job                 : Ptr32 _EJOB
+0x138 SectionObject       : Ptr32 Void
+0x13c SectionBaseAddress  : Ptr32 Void
+0x140 QuotaBlock          : Ptr32 _EPROCESS_QUOTA_BLOCK
+0x144 WorkingSetWatch     : Ptr32 _PAGEFAULT_HISTORY
+0x148 Win32WindowStation  : Ptr32 Void
+0x14c InheritedFromUniqueProcessId : Ptr32 Void
+0x150 LdtInformation      : Ptr32 Void
+0x154 VadFreeHint         : Ptr32 Void
+0x158 VdmObjects          : Ptr32 Void
+0x15c DeviceMap           : Ptr32 Void
+0x160 PhysicalVadList     : _LIST_ENTRY
+0x168 PageDirectoryPte    : _HARDWARE_PTE
+0x168 Filler              : UInt8B
+0x170 Session             : Ptr32 Void
+0x174 ImageFileName       : [16] UChar
+0x184 JobLinks            : _LIST_ENTRY
+0x18c LockedPagesList     : Ptr32 Void
+0x190 ThreadListHead      : _LIST_ENTRY
+0x198 SecurityPort        : Ptr32 Void
+0x19c PaeTop              : Ptr32 Void
+0x1a0 ActiveThreads       : UInt4B
+0x1a4 GrantedAccess       : UInt4B
+0x1a8 DefaultHardErrorProcessing : UInt4B
+0x1ac LastThreadExitStatus : Int4B
+0x1b0 Peb                 : Ptr32 _PEB
+0x1b4 PrefetchTrace       : _EX_FAST_REF
+0x1b8 ReadOperationCount  : _LARGE_INTEGER
+0x1c0 WriteOperationCount : _LARGE_INTEGER
+0x1c8 OtherOperationCount : _LARGE_INTEGER
+0x1d0 ReadTransferCount   : _LARGE_INTEGER
+0x1d8 WriteTransferCount  : _LARGE_INTEGER
+0x1e0 OtherTransferCount  : _LARGE_INTEGER
+0x1e8 CommitChargeLimit   : UInt4B
+0x1ec CommitChargePeak    : UInt4B
+0x1f0 AweInfo             : Ptr32 Void
+0x1f4 SeAuditProcessCreationInfo : _SE_AUDIT_PROCESS_CREATION_INFO
+0x1f8 Vm                  : _MM3SUPPORT
+0x238 LastFaultCount      : UInt4B
+0x23c ModifiedPageCount   : UInt4B
+0x240 NumberOfVads        : UInt4B
+0x244 JobStatus           : UInt4B

```

```

+0x248 Flags : Uint4B
+0x248 CreateReported : Pos 0, 1 Bit
+0x248 NoDebugInherit : Pos 1, 1 Bit
+0x248 ProcessExiting : Pos 2, 1 Bit
+0x248 ProcessDelete : Pos 3, 1 Bit
+0x248 Wow64SplitPages : Pos 4, 1 Bit
+0x248 VmDeleted : Pos 5, 1 Bit
+0x248 OutswapEnabled : Pos 6, 1 Bit
+0x248 Outswapped : Pos 7, 1 Bit
+0x248 ForkFailed : Pos 8, 1 Bit
+0x248 HasPhysicalVad : Pos 9, 1 Bit
+0x248 AddressSpaceInitialized : Pos 10, 2 Bits
+0x248 SetTimerResolution : Pos 12, 1 Bit
+0x248 BreakOnTermination : Pos 13, 1 Bit
+0x248 SessionCreationUnderway : Pos 14, 1 Bit
+0x248 WriteWatch : Pos 15, 1 Bit
+0x248 ProcessInSession : Pos 16, 1 Bit
+0x248 OverrideAddressSpace : Pos 17, 1 Bit
+0x248 HasAddressSpace : Pos 18, 1 Bit
+0x248 LaunchPrefetched : Pos 19, 1 Bit
+0x248 InjectInpageErrors : Pos 20, 1 Bit
+0x248 VmTopDown : Pos 21, 1 Bit
+0x248 Unused3 : Pos 22, 1 Bit
+0x248 Unused4 : Pos 23, 1 Bit
+0x248 VdmAllowed : Pos 24, 1 Bit
+0x248 Unused : Pos 25, 5 Bits
+0x248 Unused1 : Pos 30, 1 Bit
+0x248 Unused2 : Pos 31, 1 Bit
+0x24c ExitStatus : Int4B
+0x250 NextPageColor : Uint2B
+0x252 SubSystemMinorVersion : UChar
+0x253 SubSystemMajorVersion : UChar
+0x252 SubSystemVersion : Uint2B
+0x254 PriorityClass : UChar
+0x255 WorkingSetAcquiredUnsafe : UChar
+0x258 Cookie : Uint4B
kd> !process
PROCESS 89202af8 SessionId: 0 Cid: 0e9c Peb: 7ffdc000 ParentCid: 05e0
DirBase: 0ca8a000 ObjectTable: e15c3eb0 HandleCount: 126
Image: kd.exe
VadRoot 89bb4e40 Vads 51 Clone 0 Private 1744. Modified 5. Locked 0.
DeviceMap e14f75b8
Token e5d4d7f0
ElapsedTime 00:00:02.663
UserTime 00:00:00.310
KernelTime 00:00:00.060
QuotaPoolUsage[PagedPool] 15680
QuotaPoolUsage[NonPagedPool] 2040
Working Set Sizes (now,min,max) (2348, 50, 345) (9392KB, 200KB, 1380KB)
PeakWorkingSetSize 2348
VirtualSize 21 Mb
PeakVirtualSize 21 Mb
PageFaultCount 3551
MemoryPriority BACKGROUND
BasePriority 8
CommitCharge 2076

THREAD 88493668 Cid 0e9c.0ad4 Teb: 7ffdf000 Win32Thread: 00000000 RUNN
ING on processor 0
THREAD 88ffdda8 Cid 0e9c.0a5c Teb: 7ffde000 Win32Thread: 00000000 WAIT
: (WrLpcReply) UserMode Non-Alertable
88ffdf9c Semaphore Limit 0x1

kd>

```

```

kd> dt -a -b -v _EPROCESS
struct _EPROCESS, 107 elements, 0x260 bytes
+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
+0x018 DirectoryTableBase : (2 elements) UInt4B
+0x020 LdtDescriptor : struct _KGDTENTRY, 3 elements, 0x8 bytes
+0x000 LimitLow : UInt2B
+0x002 BaseLow : UInt2B
+0x004 HighWord : union __unnamed, 2 elements, 0x4 bytes
+0x000 Bytes : struct __unnamed, 4 elements, 0x4 bytes
+0x000 BaseMid : UChar
+0x001 Flags1 : UChar
+0x002 Flags2 : UChar
+0x003 BaseHi : UChar
+0x000 Bits : struct __unnamed, 10 elements, 0x4 bytes
+0x000 BaseMid : Bitfield Pos 0, 8 Bits
+0x000 Type : Bitfield Pos 8, 5 Bits
+0x000 Dpl : Bitfield Pos 13, 2 Bits
+0x000 Pres : Bitfield Pos 15, 1 Bit
+0x000 LimitHi : Bitfield Pos 16, 4 Bits
+0x000 Sys : Bitfield Pos 20, 1 Bit
+0x000 Reserved_0 : Bitfield Pos 21, 1 Bit
+0x000 Default_Big : Bitfield Pos 22, 1 Bit
+0x000 Granularity : Bitfield Pos 23, 1 Bit
+0x000 BaseHi : Bitfield Pos 24, 8 Bits
+0x028 Int21Descriptor : struct _KIDTENTRY, 4 elements, 0x8 bytes
+0x000 Offset : UInt2B
+0x002 Selector : UInt2B
+0x004 Access : UInt2B
+0x006 ExtendedOffset : UInt2B
+0x030 IopmOffset : UInt2B
+0x032 Iopl : UChar
+0x033 Unused : UChar
+0x034 ActiveProcessors : UInt4B
+0x038 KernelTime : UInt4B
+0x03c UserTime : UInt4B
+0x040 ReadyListHead : struct _LIST_ENTRY, 2 elements, 0x8 bytes
+0x000 Flink : Ptr32 to
+0x004 Blink : Ptr32 to
+0x048 SwapListEntry : struct _SINGLE_LIST_ENTRY, 1 elements, 0x4 bytes
+0x000 Next : Ptr32 to
+0x04c VdmTrapHandler : Ptr32 to
+0x050 ThreadListHead : struct _LIST_ENTRY, 2 elements, 0x8 bytes
+0x000 Flink : Ptr32 to
+0x004 Blink : Ptr32 to
+0x058 ProcessLock : UInt4B
+0x05c Affinity : UInt4B
+0x060 StackCount : UInt2B
+0x062 BasePriority : Char
+0x063 ThreadQuantum : Char
+0x064 AutoAlignment : UChar
+0x065 State : UChar
+0x066 ThreadSeed : UChar

```

```

+0x067 DisableBoost      : UChar
+0x068 PowerState       : UChar
+0x069 DisableQuantum   : UChar
+0x06a IdealNode        : UChar
+0x06b Flags            : struct _KEXECUTE_OPTIONS, 7 elements, 0x1 bytes
    +0x000 ExecuteDisable : Bitfield Pos 0, 1 Bit
    +0x000 ExecuteEnable  : Bitfield Pos 1, 1 Bit
    +0x000 DisableThunkEmulation : Bitfield Pos 2, 1 Bit
    +0x000 Permanent      : Bitfield Pos 3, 1 Bit
    +0x000 ExecuteDispatchEnable : Bitfield Pos 4, 1 Bit
    +0x000 ImageDispatchEnable : Bitfield Pos 5, 1 Bit
    +0x000 Spare          : Bitfield Pos 6, 2 Bits
+0x06b ExecuteOptions    : UChar
+0x06c ProcessLock       : struct _EX_PUSH_LOCK, 5 elements, 0x4 bytes
    +0x000 Waiting        : Bitfield Pos 0, 1 Bit
    +0x000 Exclusive      : Bitfield Pos 1, 1 Bit
    +0x000 Shared         : Bitfield Pos 2, 30 Bits
    +0x000 Value          : Uint4B
    +0x000 Ptr            : Ptr32 to
+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
+0x080 RundownProtect    : struct _EX_RUNDOWN_REF, 2 elements, 0x4 bytes
    +0x000 Count          : Uint4B
    +0x000 Ptr            : Ptr32 to
+0x084 UniqueProcessId   : Ptr32 to
+0x088 ActiveProcessLinks : struct _LIST_ENTRY, 2 elements, 0x8 bytes
    +0x000 Flink          : Ptr32 to
    +0x004 Blink          : Ptr32 to
+0x090 QuotaUsage         : (3 elements) Uint4B
+0x09c QuotaPeak          : (3 elements) Uint4B
+0x0a8 CommitCharge       : Uint4B
+0x0ac PeakVirtualSize    : Uint4B
+0x0b0 VirtualSize        : Uint4B
+0x0b4 SessionProcessLinks : struct _LIST_ENTRY, 2 elements, 0x8 bytes
    +0x000 Flink          : Ptr32 to
    +0x004 Blink          : Ptr32 to
+0x0bc DebugPort         : Ptr32 to
+0x0c0 ExceptionPort      : Ptr32 to
+0x0c4 ObjectTable        : Ptr32 to
+0x0c8 Token              : struct _EX_FAST_REF, 3 elements, 0x4 bytes
    +0x000 Object          : Ptr32 to
    +0x000 RefCnt          : Bitfield Pos 0, 3 Bits
    +0x000 Value           : Uint4B
+0x0cc WorkingSetLock     : struct _FAST_MUTEX, 5 elements, 0x20 bytes
    +0x000 Count           : Int4B
    +0x004 Owner           : Ptr32 to
    +0x008 Contention      : Uint4B
    +0x00c Event           : struct _KEVENT, 1 elements, 0x10 bytes
    +0x000 Header          : struct _DISPATCHER_HEADER, 6 elements, 0x10 bytes
ytes
    +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
        +0x01c OldIrql            : Uint4B
+0x0ec WorkingSetPage           : Uint4B
+0x0f0 AddressCreationLock      : struct _FAST_MUTEX, 5 elements, 0x20 bytes
            +0x000 Count          : Int4B
            +0x004 Owner          : Ptr32 to
            +0x008 Contention     : Uint4B
            +0x00c Event          : struct _KEVENT, 1 elements, 0x10 bytes
            +0x000 Header         : struct _DISPATCHER_HEADER, 6 elements, 0x10 b
ytes
            +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
        +0x01c OldIrql            : Uint4B
+0x110 HyperSpaceLock           : Uint4B
+0x114 ForkInProgress           : Ptr32 to
+0x118 HardwareTrigger          : Uint4B
+0x11c VadRoot                  : Ptr32 to
+0x120 VadHint                  : Ptr32 to
+0x124 CloneRoot                : Ptr32 to
+0x128 NumberOfPrivatePages     : Uint4B
+0x12c NumberOfLockedPages      : Uint4B
+0x130 Win32Process              : Ptr32 to
+0x134 Job                      : Ptr32 to
+0x138 SectionObject            : Ptr32 to
+0x13c SectionBaseAddress       : Ptr32 to
+0x140 QuotaBlock               : Ptr32 to
+0x144 WorkingSetWatch          : Ptr32 to
+0x148 Win32WindowStation       : Ptr32 to
+0x14c InheritedFromUniqueProcessId : Ptr32 to
+0x150 LdtInformation           : Ptr32 to
+0x154 VadFreeHint              : Ptr32 to
+0x158 VdmObjects               : Ptr32 to
+0x15c DeviceMap                : Ptr32 to
+0x160 PhysicalVadList          : struct _LIST_ENTRY, 2 elements, 0x8 bytes
            +0x000 Flink          : Ptr32 to
            +0x004 Blink          : Ptr32 to
+0x168 PageDirectoryPte        : struct _HARDWARE_PTE_X86, 13 elements, 0x4 bytes
            +0x000 Valid          : Bitfield Pos 0, 1 Bit
            +0x000 Write          : Bitfield Pos 1, 1 Bit
            +0x000 Owner          : Bitfield Pos 2, 1 Bit
            +0x000 WriteThrough   : Bitfield Pos 3, 1 Bit
            +0x000 CacheDisable   : Bitfield Pos 4, 1 Bit
            +0x000 Accessed       : Bitfield Pos 5, 1 Bit
            +0x000 Dirty          : Bitfield Pos 6, 1 Bit
            +0x000 LargePage      : Bitfield Pos 7, 1 Bit
            +0x000 Global         : Bitfield Pos 8, 1 Bit
            +0x000 CopyOnWrite    : Bitfield Pos 9, 1 Bit
            +0x000 Prototype      : Bitfield Pos 10, 1 Bit
            +0x000 reserved       : Bitfield Pos 11, 1 Bit
            +0x000 PageFrameNumber : Bitfield Pos 12, 20 Bits
+0x168 Filler                   : Uint8B
+0x170 Session                  : Ptr32 to
+0x174 ImageFileName            : (16 elements) UChar
+0x184 JobLinks                  : struct _LIST_ENTRY, 2 elements, 0x8 bytes
            +0x000 Flink          : Ptr32 to
            +0x004 Blink          : Ptr32 to
+0x18c LockedPagesList          : Ptr32 to
+0x190 ThreadListHead           : struct _LIST_ENTRY, 2 elements, 0x8 bytes
            +0x000 Flink          : Ptr32 to

```



```

+0x004 Blink : Ptr32 to
+0x198 SecurityPort : Ptr32 to
+0x19c PaeTop : Ptr32 to
+0x1a0 ActiveThreads : Uint4B
+0x1a4 GrantedAccess : Uint4B
+0x1a8 DefaultHardErrorProcessing : Uint4B
+0x1ac LastThreadExitStatus : Int4B
+0x1b0 Peb : Ptr32 to
+0x1b4 PrefetchTrace : struct _EX_FAST_REF, 3 elements, 0x4 bytes
+0x000 Object : Ptr32 to
+0x000 RefCnt : Bitfield Pos 0, 3 Bits
+0x000 Value : Uint4B
+0x1b8 ReadOperationCount : 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
+0x1c0 WriteOperationCount : 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
+0x1c8 OtherOperationCount : 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
+0x1d0 ReadTransferCount : 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
+0x1d8 WriteTransferCount : 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
+0x1e0 OtherTransferCount : 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
+0x1e8 CommitChargeLimit : Uint4B
+0x1ec CommitChargePeak : Uint4B
+0x1f0 AweInfo : Ptr32 to
+0x1f4 SeAuditProcessCreationInfo : struct _SE_AUDIT_PROCESS_CREATION_INFO, 1
elements, 0x4 bytes
+0x000 ImageFileName : Ptr32 to
+0x1f8 Vm : struct _MMSUPPORT, 14 elements, 0x40 bytes
+0x000 LastTrimTime : 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
+0x008 Flags             : struct _MMSUPPORT_FLAGS, 9 elements, 0x4 bytes
+0x000 SessionSpace      : Bitfield Pos 0, 1 Bit
+0x000 BeingTrimmed      : Bitfield Pos 1, 1 Bit
+0x000 SessionLeader     : Bitfield Pos 2, 1 Bit
+0x000 TrimHard          : Bitfield Pos 3, 1 Bit
+0x000 WorkingSetHard    : Bitfield Pos 4, 1 Bit
+0x000 AddressSpaceBeingDeleted : Bitfield Pos 5, 1 Bit
+0x000 Available         : Bitfield Pos 6, 10 Bits
+0x000 AllowWorkingSetAdjustment : Bitfield Pos 16, 8 Bits
+0x000 MemoryPriority     : Bitfield Pos 24, 8 Bits
+0x00c PageFaultCount    : Uint4B
+0x010 PeakWorkingSetSize : Uint4B
+0x014 WorkingSetSize    : Uint4B
+0x018 MinimumWorkingSetSize : Uint4B
+0x01c MaximumWorkingSetSize : Uint4B
+0x020 VmWorkingSetList  : Ptr32 to
+0x024 WorkingSetExpansionLinks : struct _LIST_ENTRY, 2 elements, 0x8 byte

+0x000 Flink             : Ptr32 to
+0x004 Blink             : Ptr32 to
+0x02c Claim             : Uint4B
+0x030 NextEstimationSlot : Uint4B
+0x034 NextAgingSlot     : Uint4B
+0x038 EstimatedAvailable : Uint4B
+0x03c GrowthSinceLastEstimate : Uint4B
+0x238 LastFaultCount    : Uint4B
+0x23c ModifiedPageCount : Uint4B
+0x240 NumberOfVads      : Uint4B
+0x244 JobStatus         : Uint4B
+0x248 Flags             : Uint4B
+0x248 CreateReported    : Bitfield Pos 0, 1 Bit
+0x248 NoDebugInherit    : Bitfield Pos 1, 1 Bit
+0x248 ProcessExiting     : Bitfield Pos 2, 1 Bit
+0x248 ProcessDelete    : Bitfield Pos 3, 1 Bit
+0x248 Wow64SplitPages    : Bitfield Pos 4, 1 Bit
+0x248 VmDeleted          : Bitfield Pos 5, 1 Bit
+0x248 OutswapEnabled     : Bitfield Pos 6, 1 Bit
+0x248 Outswapped         : Bitfield Pos 7, 1 Bit
+0x248 ForkFailed         : Bitfield Pos 8, 1 Bit
+0x248 HasPhysicalVad     : Bitfield Pos 9, 1 Bit
+0x248 AddressSpaceInitialized : Bitfield Pos 10, 2 Bits
+0x248 SetTimerResolution : Bitfield Pos 12, 1 Bit
+0x248 BreakOnTermination : Bitfield Pos 13, 1 Bit
+0x248 SessionCreationUnderway : Bitfield Pos 14, 1 Bit
+0x248 WriteWatch        : Bitfield Pos 15, 1 Bit
+0x248 ProcessInSession   : Bitfield Pos 16, 1 Bit
+0x248 OverrideAddressSpace : Bitfield Pos 17, 1 Bit
+0x248 HasAddressSpace     : Bitfield Pos 18, 1 Bit
+0x248 LaunchPrefetched   : Bitfield Pos 19, 1 Bit
+0x248 InjectInpageErrors : Bitfield Pos 20, 1 Bit
+0x248 VmTopDown          : Bitfield Pos 21, 1 Bit
+0x248 Unused3            : Bitfield Pos 22, 1 Bit
+0x248 Unused4            : Bitfield Pos 23, 1 Bit
+0x248 VdmAllowed         : Bitfield Pos 24, 1 Bit
+0x248 Unused             : Bitfield Pos 25, 5 Bits
+0x248 Unused1            : Bitfield Pos 30, 1 Bit
+0x248 Unused2            : Bitfield Pos 31, 1 Bit
+0x24c ExitStatus         : Int4B
+0x250 NextPageColor      : Uint2B
+0x252 SubSystemMinorVersion : UChar
+0x253 SubSystemMajorVersion : UChar
+0x252 SubSystemVersion    : Uint2B
+0x254 PriorityClass       : UChar
+0x255 WorkingSetAcquiredUnsafe : UChar
+0x258 Cookie             : Uint4B

```

Appendix C: EPROCESS structure of Microsoft Windows 2000, Service Pack 4.

```

kd> dt -a -b -v _EPROCESS
struct _EPROCESS, 94 elements, 0x290 bytes
    +0x000 Pcb                : struct _KPROCESS, 26 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
    +0x018 DirectoryTableBase : (2 elements) Uint4B
    +0x020 LdtDescriptor      : struct _KGDTENTRY, 3 elements, 0x8 bytes
        +0x000 LimitLow       : Uint2B
        +0x002 BaseLow        : Uint2B
        +0x004 HighWord       : union __unnamed, 2 elements, 0x4 bytes
            +0x000 Bytes      : struct __unnamed, 4 elements, 0x4 bytes
                +0x000 BaseMid : UChar
                +0x001 Flags1   : UChar
                +0x002 Flags2   : UChar
                +0x003 BaseHi   : UChar
            +0x000 Bits        : struct __unnamed, 10 elements, 0x4 bytes
                +0x000 BaseMid : Bitfield Pos 0, 8 Bits
                +0x000 Type    : Bitfield Pos 8, 5 Bits
                +0x000 Dpl     : Bitfield Pos 13, 2 Bits
                +0x000 Pres    : Bitfield Pos 15, 1 Bit
                +0x000 LimitHi : Bitfield Pos 16, 4 Bits
                +0x000 Sys     : Bitfield Pos 20, 1 Bit
                +0x000 Reserved_0 : Bitfield Pos 21, 1 Bit
                +0x000 Default_Big : Bitfield Pos 22, 1 Bit
                +0x000 Granularity : Bitfield Pos 23, 1 Bit
                +0x000 BaseHi   : Bitfield Pos 24, 8 Bits
    +0x028 Int21Descriptor    : struct _KIDTENTRY, 4 elements, 0x8 bytes
        +0x000 Offset        : Uint2B
        +0x002 Selector      : Uint2B
        +0x004 Access        : Uint2B
        +0x006 ExtendedOffset : Uint2B
    +0x030 IopmOffset         : Uint2B
    +0x032 Iopl               : UChar
    +0x033 VdmFlag            : UChar
    +0x034 ActiveProcessors   : Uint4B
    +0x038 KernelTime         : Uint4B
    +0x03c UserTime           : Uint4B
    +0x040 ReadyListHead     : struct _LIST_ENTRY, 2 elements, 0x8 bytes
        +0x000 Flink          : Ptr32 to
        +0x004 Blink          : Ptr32 to
    +0x048 SwapListEntry      : struct _LIST_ENTRY, 2 elements, 0x8 bytes
        +0x000 Flink          : Ptr32 to
        +0x004 Blink          : Ptr32 to
    +0x050 ThreadListHead     : struct _LIST_ENTRY, 2 elements, 0x8 bytes

```

```

+0x000 Flink          : Ptr32 to
+0x004 Blink          : Ptr32 to
+0x058 ProcessLock    : Uint4B
+0x05c Affinity        : Uint4B
+0x060 StackCount      : Uint2B
+0x062 BasePriority     : Char
+0x063 ThreadQuantum   : Char
+0x064 AutoAlignment   : UChar
+0x065 State           : UChar
+0x066 ThreadSeed      : UChar
+0x067 DisableBoost    : UChar
+0x068 PowerState      : UChar
+0x069 DisableQuantum  : UChar
+0x06a Spare           : (2 elements) UChar
+0x06c ExitStatus      : Int4B
+0x070 LockEvent       : struct _KEVENT, 1 elements, 0x10 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
+0x080 LockCount       : Uint4B
+0x088 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
+0x090 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
+0x098 LockOwner       : Ptr32 to
+0x09c UniqueProcessId : Ptr32 to
+0x0a0 ActiveProcessLinks : struct _LIST_ENTRY, 2 elements, 0x8 bytes
+0x000 Flink          : Ptr32 to
+0x004 Blink          : Ptr32 to
+0x0a8 QuotaPeakPoolUsage : (2 elements) Uint4B
+0x0b0 QuotaPoolUsage   : (2 elements) Uint4B
+0x0b8 PagefileUsage    : Uint4B
+0x0bc CommitCharge     : Uint4B
+0x0c0 PeakPagefileUsage : Uint4B
+0x0c4 PeakVirtualSize  : Uint4B
+0x0c8 VirtualSize      : Uint4B
+0x0d0 Vm               : struct _MMSUPPORT, 19 elements, 0x48 bytes
+0x000 LastTrimTime    : 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
+0x008 LastTrimFaultCount : Uint4B
+0x00c PageFaultCount    : Uint4B
+0x010 PeakWorkingSetSize : Uint4B
+0x014 WorkingSetSize    : Uint4B
+0x018 MinimumWorkingSetSize : Uint4B
+0x01c MaximumWorkingSetSize : Uint4B
+0x020 VmWorkingSetList  : Ptr32 to
+0x024 WorkingSetExpansionLinks : struct _LIST_ENTRY, 2 elements, 0x8
bytes
+0x000 Flink              : Ptr32 to
+0x004 Blink              : Ptr32 to
+0x02c AllowWorkingSetAdjustment : UChar
+0x02d AddressSpaceBeingDeleted : UChar
+0x02e ForegroundSwitchCount : UChar
+0x02f MemoryPriority      : UChar
+0x030 u                  : union __unnamed, 2 elements, 0x4 bytes
+0x000 LongFlags          : Uint4B
+0x000 Flags              : struct _MMSUPPORT_FLAGS, 8 elements, 0x4
bytes
+0x000 SessionSpace      : Bitfield Pos 0, 1 Bit
+0x000 BeingTrimmed      : Bitfield Pos 1, 1 Bit
+0x000 ProcessInSession  : Bitfield Pos 2, 1 Bit
+0x000 SessionLeader     : Bitfield Pos 3, 1 Bit
+0x000 TrimHard          : Bitfield Pos 4, 1 Bit
+0x000 WorkingSetHard    : Bitfield Pos 5, 1 Bit
+0x000 WriteWatch        : Bitfield Pos 6, 1 Bit
+0x000 Filler            : Bitfield Pos 7, 25 Bits
+0x034 Claim              : Uint4B
+0x038 NextEstimationSlot : Uint4B
+0x03c NextAgingSlot     : Uint4B
+0x040 EstimatedAvailable : Uint4B
+0x044 GrowthSinceLastEstimate : Uint4B
+0x118 SessionProcessLinks : struct _LIST_ENTRY, 2 elements, 0x8 bytes
+0x000 Flink              : Ptr32 to
+0x004 Blink              : Ptr32 to
+0x120 DebugPort          : Ptr32 to
+0x124 ExceptionPort      : Ptr32 to
+0x128 ObjectTable        : Ptr32 to
+0x12c Token              : Ptr32 to
+0x130 WorkingSetLock     : struct _FAST_MUTEX, 5 elements, 0x20 bytes
+0x000 Count              : Int4B
+0x004 Owner              : Ptr32 to
+0x008 Contention         : Uint4B
+0x00c Event              : struct _KEVENT, 1 elements, 0x10 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
+0x01c OldIrql            : Uint4B
+0x150 WorkingSetPage     : Uint4B
+0x154 ProcessOutswapEnabled : UChar
+0x155 ProcessOutswapped : UChar
+0x156 AddressSpaceInitialized : UChar

```

```

+0x157 AddressSpaceDeleted : UChar
+0x158 AddressCreationLock : struct _FAST_MUTEX, 5 elements, 0x20 bytes
    +0x000 Count : Int4B
    +0x004 Owner : Ptr32 to
    +0x008 Contention : Uint4B
    +0x00c Event : struct _KEVENT, 1 elements, 0x10 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
    +0x01c OldIrql : Uint4B
+0x178 HyperSpaceLock : Uint4B
+0x17c ForkInProgress : Ptr32 to
+0x180 VmOperation : Uint2B
+0x182 ForkWasSuccessful : UChar
+0x183 MmAggressiveWsTrimMask : UChar
+0x184 VmOperationEvent : Ptr32 to
+0x188 PaeTop : Ptr32 to
+0x18c LastFaultCount : Uint4B
+0x190 ModifiedPageCount : Uint4B
+0x194 VadRoot : Ptr32 to
+0x198 VadHint : Ptr32 to
+0x19c CloneRoot : Ptr32 to
+0x1a0 NumberOfPrivatePages : Uint4B
+0x1a4 NumberOfLockedPages : Uint4B
+0x1a8 NextPageColor : Uint2B
+0x1aa ExitProcessCalled : UChar
+0x1ab CreateProcessReported : UChar
+0x1ac SectionHandle : Ptr32 to
+0x1b0 Peb : Ptr32 to
+0x1b4 SectionBaseAddress : Ptr32 to
+0x1b8 QuotaBlock : Ptr32 to
+0x1bc LastThreadExitStatus : Int4B
+0x1c0 WorkingSetWatch : Ptr32 to
+0x1c4 Win32WindowStation : Ptr32 to
+0x1c8 InheritedFromUniqueProcessId : Ptr32 to
+0x1cc GrantedAccess : Uint4B
+0x1d0 DefaultHardErrorProcessing : Uint4B
+0x1d4 LdtInformation : Ptr32 to
+0x1d8 VadFreeHint : Ptr32 to
+0x1dc VdmObjects : Ptr32 to
+0x1e0 DeviceMap : Ptr32 to
+0x1e4 SessionId : Uint4B
+0x1e8 PhysicalVadList : struct _LIST_ENTRY, 2 elements, 0x8 bytes
    +0x000 Flink : Ptr32 to
    +0x004 Blink : Ptr32 to
+0x1f0 PageDirectoryPte : struct _HARDWARE_PTE_X86, 13 elements, 0x4 bytes
    +0x000 Valid : Bitfield Pos 0, 1 Bit
    +0x000 Write : Bitfield Pos 1, 1 Bit
    +0x000 Owner : Bitfield Pos 2, 1 Bit
    +0x000 WriteThrough : Bitfield Pos 3, 1 Bit
    +0x000 CacheDisable : Bitfield Pos 4, 1 Bit
    +0x000 Accessed : Bitfield Pos 5, 1 Bit
    +0x000 Dirty : Bitfield Pos 6, 1 Bit

```

```

+0x000 LargePage          : Bitfield Pos 7, 1 Bit
+0x000 Global             : Bitfield Pos 8, 1 Bit
+0x000 CopyOnWrite        : Bitfield Pos 9, 1 Bit
+0x000 Prototype          : Bitfield Pos 10, 1 Bit
+0x000 reserved           : Bitfield Pos 11, 1 Bit
+0x000 PageFrameNumber    : Bitfield Pos 12, 20 Bits
+0x1f0 Filler              : Uint8B
+0x1f8 PaePageDirectoryPage : Uint4B
+0x1fc ImageFileName       : (16 elements)  UChar
+0x20c VmTrimFaultValue   : Uint4B
+0x210 SetTimerResolution : UChar
+0x211 PriorityClass       : UChar
+0x212 SubSystemMinorVersion : UChar
+0x213 SubSystemMajorVersion : UChar
+0x212 SubSystemVersion    : Uint2B
+0x214 Win32Process        : Ptr32 to
+0x218 Job                 : Ptr32 to
+0x21c JobStatus           : Uint4B
+0x220 JobLinks            : struct _LIST_ENTRY, 2 elements, 0x8 bytes
    +0x000 Flink            : Ptr32 to
    +0x004 Blink            : Ptr32 to
+0x228 LockedPagesList    : Ptr32 to
+0x22c SecurityPort        : Ptr32 to
+0x230 Wow64Process        : Ptr32 to
+0x238 ReadOperationCount  : 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
+0x240 WriteOperationCount : 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
+0x248 OtherOperationCount : 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
+0x250 ReadTransferCount   : 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
+0x258 WriteTransferCount  : 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
+0x260 OtherTransferCount  : 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
+0x268 CommitChargeLimit : Uint4B
+0x26c CommitChargePeak : Uint4B
+0x270 ThreadListHead   : struct _LIST_ENTRY, 2 elements, 0x8 bytes
    +0x000 Flink        : Ptr32 to
    +0x004 Blink        : Ptr32 to
+0x278 VadPhysicalPagesBitMap : Ptr32 to
+0x27c VadPhysicalPages : Uint4B
+0x280 AweLock          : Uint4B
+0x284 pImageFileName   : Ptr32 to
+0x288 Session          : Ptr32 to
+0x28c Flags            : Uint4B

```


Appendix D: ETHREAD structure dump from WinXP SP2

```

kd> dt -a -b -v _ETHREAD
struct _ETHREAD, 54 elements, 0x258 bytes
    +0x000 Tcb                : struct _KTHREAD, 73 elements, 0x1c0 bytes
    +0x000 Header              : struct _DISPATCHER_HEADER, 6 elements, 0x10
byte
s
    +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 MutantListHead      : struct _LIST_ENTRY, 2 elements, 0x8 bytes
        +0x000 Flink           : Ptr32 to
        +0x004 Blink           : Ptr32 to
    +0x018 InitialStack        : Ptr32 to
    +0x01c StackLimit          : Ptr32 to
    +0x020 Teb                 : Ptr32 to
    +0x024 TlsArray             : Ptr32 to
    +0x028 KernelStack          : Ptr32 to
    +0x02c DebugActive          : UChar
    +0x02d State                : UChar
    +0x02e Alerted              : (2 elements) UChar
    +0x030 Iopl                 : UChar
    +0x031 NpxState             : UChar
    +0x032 Saturation           : Char
    +0x033 Priority             : Char
    +0x034 ApcState             : struct _KAPC_STATE, 5 elements, 0x18 bytes
    +0x000 ApcListHead         : (2 elements) struct _LIST_ENTRY, 2
elements,
0x8 bytes
        +0x000 Flink           : Ptr32 to
        +0x004 Blink           : Ptr32 to
    +0x010 Process              : Ptr32 to
    +0x014 KernelApcInProgress : UChar
    +0x015 KernelApcPending    : UChar
    +0x016 UserApcPending       : UChar
    +0x04c ContextSwitches      : Uint4B
    +0x050 IdleSwapBlock        : UChar
    +0x051 Spare0               : (3 elements) UChar
    +0x054 WaitStatus           : Int4B
    +0x058 WaitIrql             : UChar
    +0x059 WaitMode             : Char
    +0x05a WaitNext             : UChar
    +0x05b WaitReason           : UChar
    +0x05c WaitBlockList        : Ptr32 to
    +0x060 WaitListEntry        : struct _LIST_ENTRY, 2 elements, 0x8 bytes
        +0x000 Flink           : Ptr32 to
        +0x004 Blink           : Ptr32 to
    +0x060 SwapListEntry        : struct _SINGLE_LIST_ENTRY, 1 elements, 0x4
bytes
        +0x000 Next            : Ptr32 to

```

```

+0x068 WaitTime      : Uint4B
+0x06c BasePriority   : Char
+0x06d DecrementCount : UChar
+0x06e PriorityDecrement : Char
+0x06f Quantum       : Char
+0x070 WaitBlock     : (4 elements) struct _KWAIT_BLOCK, 6 elements,
0
x18 bytes
+0x000 WaitListEntry : struct _LIST_ENTRY, 2 elements, 0x8 bytes
+0x000 Flink         : Ptr32 to
+0x004 Blink         : Ptr32 to
+0x008 Thread        : Ptr32 to
+0x00c Object        : Ptr32 to
+0x010 NextWaitBlock : Ptr32 to
+0x014 WaitKey       : Uint2B
+0x016 WaitType      : Uint2B
+0x0d0 LegoData      : Ptr32 to
+0x0d4 KernelApcDisable : Uint4B
+0x0d8 UserAffinity  : Uint4B
+0x0dc SystemAffinityActive : UChar
+0x0dd PowerState    : UChar
+0x0de NpxIrql       : UChar
+0x0df InitialNode   : UChar
+0x0e0 ServiceTable  : Ptr32 to
+0x0e4 Queue         : Ptr32 to
+0x0e8 ApcQueueLock  : Uint4B
+0x0f0 Timer         : struct _KTIMER, 5 elements, 0x28 bytes
+0x000 Header        : struct _DISPATCHER_HEADER, 6 elements, 0x10
b
ytes
+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 DueTime       : union _ULARGE_INTEGER, 4 elements, 0x8 bytes
+0x000 LowPart       : Uint4B
+0x004 HighPart      : Uint4B
+0x000 u             : struct __unnamed, 2 elements, 0x8 bytes
+0x000 LowPart       : Uint4B
+0x004 HighPart      : Uint4B
+0x000 QuadPart      : Uint8B
+0x018 TimerListEntry : struct _LIST_ENTRY, 2 elements, 0x8 bytes
+0x000 Flink         : Ptr32 to
+0x004 Blink         : Ptr32 to
+0x020 Dpc           : Ptr32 to
+0x024 Period        : Int4B
+0x118 QueueListEntry : struct _LIST_ENTRY, 2 elements, 0x8 bytes
+0x000 Flink         : Ptr32 to
+0x004 Blink         : Ptr32 to
+0x120 SoftAffinity  : Uint4B
+0x124 Affinity       : Uint4B
+0x128 Prccmpted     : UChar
+0x129 ProcessReadyQueue : UChar
+0x12a KernelStackResident : UChar
+0x12b NextProcessor : UChar
+0x12c CallbackStack : Ptr32 to

```

```

+0x130 Win32Thread      : Ptr32 to
+0x134 TrapFrame        : Ptr32 to
+0x138 ApcStatePointer  : (2 elements) Ptr32 to
+0x140 PreviousMode     : Char
+0x141 EnableStackSwap  : UChar
+0x142 LargeStack       : UChar
+0x143 ResourceIndex    : UChar
+0x144 KernelTime       : Uint4B
+0x148 UserTime         : Uint4B
+0x14c SavedApcState    : struct _KAPC_STATE, 5 elements, 0x18 bytes
+0x000 ApcListHead      : (2 elements) struct _LIST_ENTRY, 2
elements,
0x8 bytes
+0x000 Flink            : Ptr32 to
+0x004 Blink            : Ptr32 to
+0x010 Process          : Ptr32 to
+0x014 KernelApcInProgress : UChar
+0x015 KernelApcPending : UChar
+0x016 UserApcPending   : UChar
+0x164 Alertable        : UChar
+0x165 ApcStateIndex    : UChar
+0x166 ApcQueueable     : UChar
+0x167 AutoAlignment    : UChar
+0x168 StackBase        : Ptr32 to
+0x16c SuspendApc       : struct _KAPC, 14 elements, 0x30 bytes
+0x000 Type             : Int2B
+0x002 Size             : Int2B
+0x004 Spare0           : Uint4B
+0x008 Thread           : Ptr32 to
+0x00c ApcListEntry     : struct _LIST_ENTRY, 2 elements, 0x8 bytes
+0x000 Flink            : Ptr32 to
+0x004 Blink            : Ptr32 to
+0x014 KernelRoutine    : Ptr32 to
+0x018 RundownRoutine   : Ptr32 to
+0x01c NormalRoutine    : Ptr32 to
+0x020 NormalContext    : Ptr32 to
+0x024 SystemArgument1  : Ptr32 to
+0x028 SystemArgument2  : Ptr32 to
+0x02c ApcStateIndex    : Char
+0x02d ApcMode          : Char
+0x02e Inserted         : UChar
+0x19c SuspendSemaphore : struct _KSEMAPHORE, 2 elements, 0x14 bytes
+0x000 Header           : struct _DISPATCHER_HEADER, 6 elements, 0x10
b
ytes
+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 Limit            : Int4B
+0x1b0 ThreadListEntry  : struct _LIST_ENTRY, 2 elements, 0x8 bytes
+0x000 Flink            : Ptr32 to
+0x004 Blink            : Ptr32 to
+0x1b8 FreezeCount      : Char
+0x1b9 SuspendCount     : Char
+0x1ba IdealProcessor   : UChar

```

```

+0x1bb DisableBoost      : UChar
+0x1c0 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
+0x1c0 NestedFaultCount  : Bitfield Pos 0, 2 Bits
+0x1c0 ApcNeeded          : Bitfield Pos 2, 1 Bit
+0x1c8 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
+0x1c8 LpcReplyChain     : struct _LIST_ENTRY, 2 elements, 0x8 bytes
    +0x000 Flink         : Ptr32 to
    +0x004 Blink         : Ptr32 to
+0x1c8 KeyedWaitChain    : struct _LIST_ENTRY, 2 elements, 0x8 bytes
    +0x000 Flink         : Ptr32 to
    +0x004 Blink         : Ptr32 to
+0x1d0 ExitStatus        : Int4B
+0x1d0 OfsChain          : Ptr32 to
+0x1d4 PostBlockList     : struct _LIST_ENTRY, 2 elements, 0x8 bytes
    +0x000 Flink         : Ptr32 to
    +0x004 Blink         : Ptr32 to
+0x1dc TerminationPort   : Ptr32 to
+0x1dc ReaperLink        : Ptr32 to
+0x1dc KeyedWaitValue    : Ptr32 to
+0x1e0 ActiveTimerListLock : UInt4B
+0x1e4 ActiveTimerListHead : struct _LIST_ENTRY, 2 elements, 0x8 bytes
    +0x000 Flink         : Ptr32 to
    +0x004 Blink         : Ptr32 to
+0x1ec Cid               : struct _CLIENT_ID, 2 elements, 0x8 bytes
    +0x000 UniqueProcess  : Ptr32 to
    +0x004 UniqueThread   : Ptr32 to
+0x1f4 LpcReplySemaphore : struct _KSEMAPHORE, 2 elements, 0x14 bytes
    +0x000 Header        : struct _DISPATCHER_HEADER, 6 elements, 0x10
byte
s
    +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 Limit          : Int4B
+0x1f4 KeyedWaitSemaphore : struct _KSEMAPHORE, 2 elements, 0x14 bytes
    +0x000 Header        : struct _DISPATCHER_HEADER, 6 elements, 0x10
byte
s
    +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 Limit              : Int4B
+0x208 LpcReplyMessage    : Ptr32 to
+0x208 LpcWaitingOnPort   : Ptr32 to
+0x20c ImpersonationInfo  : Ptr32 to
+0x210 IrpList            : struct _LIST_ENTRY, 2 elements, 0x8 bytes
+0x000 Flink              : Ptr32 to
+0x004 Blink              : Ptr32 to
+0x218 TopLevelIrp        : Uint4B
+0x21c DeviceToVerify     : Ptr32 to
+0x220 ThreadsProcess     : Ptr32 to
+0x224 StartAddress       : Ptr32 to
+0x228 Win32StartAddress  : Ptr32 to
+0x228 LpcReceivedMessageId : Uint4B
+0x22c ThreadListEntry    : struct _LIST_ENTRY, 2 elements, 0x8 bytes
+0x000 Flink              : Ptr32 to
+0x004 Blink              : Ptr32 to
+0x234 RundownProtect     : struct _EX_RUNDOWN_REF, 2 elements, 0x4 bytes
+0x000 Count              : Uint4B
+0x000 Ptr                : Ptr32 to
+0x238 ThreadLock         : struct _EX_PUSH_LOCK, 5 elements, 0x4 bytes
+0x000 Waiting            : Bitfield Pos 0, 1 Bit
+0x000 Exclusive          : Bitfield Pos 1, 1 Bit
+0x000 Shared             : Bitfield Pos 2, 30 Bits
+0x000 Value              : Uint4B
+0x000 Ptr                : Ptr32 to
+0x23c LpcReplyMessageId  : Uint4B
+0x240 ReadClusterSize    : Uint4B
+0x244 GrantedAccess      : Uint4B
+0x248 CrossThreadFlags   : Uint4B
+0x248 Terminated       : Bitfield Pos 0, 1 Bit
+0x248 DeadThread        : Bitfield Pos 1, 1 Bit
+0x248 HideFromDebugger   : Bitfield Pos 2, 1 Bit
+0x248 ActiveImpersonationInfo : Bitfield Pos 3, 1 Bit
+0x248 SystemThread       : Bitfield Pos 4, 1 Bit
+0x248 HardErrorsAreDisabled : Bitfield Pos 5, 1 Bit
+0x248 BreakOnTermination : Bitfield Pos 6, 1 Bit
+0x248 SkipCreationMsg    : Bitfield Pos 7, 1 Bit
+0x248 SkipTerminationMsg : Bitfield Pos 8, 1 Bit
+0x24c SameThreadPassiveFlags : Uint4B
+0x24c ActiveExWorker     : Bitfield Pos 0, 1 Bit
+0x24c ExWorkerCanWaitUser : Bitfield Pos 1, 1 Bit
+0x24c MemoryMaker        : Bitfield Pos 2, 1 Bit
+0x250 SameThreadApcFlags : Uint4B
+0x250 LpcReceivedMsgIdValid : Bitfield Pos 0, 1 Bit
+0x250 LpcExitThreadCalled : Bitfield Pos 1, 1 Bit
+0x250 AddressSpaceOwner  : Bitfield Pos 2, 1 Bit
+0x254 ForwardClusterOnly : UChar
+0x255 DisablePageFaultClustering : UChar
kd>

```

Appendix E: Decoding a Process Owner

One offset of an EPROCESS is the Access Token, which defines the security context for the process. It is defined as (XP SP2):

```
dt_TOKEN
+0x000 TokenSource      : _TOKEN_SOURCE
+0x010 TokenId          : _LUID
+0x018 AuthenticationId : _LUID
+0x020 ParentTokenId    : _LUID
+0x028 ExpirationTime   : _LARGE_INTEGER
+0x030 TokenLock        : Ptr32 _ERESOURCE
+0x038 AuditPolicy      : _SEP_AUDIT_POLICY
+0x040 ModifiedId       : _LUID
+0x048 SessionId        : Uint4B
+0x04c UserAndGroupCount : Uint4B
+0x050 RestrictedSidCount : Uint4B
+0x054 PrivilegeCount   : Uint4B
+0x058 VariableLength   : Uint4B
+0x05c DynamicCharged   : Uint4B
+0x060 DynamicAvailable : Uint4B
+0x064 DefaultOwnerIndex : Uint4B
+0x068 UserAndGroups     : Ptr32 _SID_AND_ATTRIBUTES
+0x06c RestrictedSids    : Ptr32 _SID_AND_ATTRIBUTES
+0x070 PrimaryGroup      : Ptr32 Void
+0x074 Privileges        : Ptr32 _LUID_AND_ATTRIBUTES
+0x078 DynamicPart       : Ptr32 Uint4B
+0x07c DefaultDacl       : Ptr32 _ACL
+0x080 TokenType         : _TOKEN_TYPE
+0x084 ImpersonationLevel : _SECURITY_IMPERSONATION_LEVEL
+0x088 TokenFlags        : Uint4B
+0x08c TokenInUse        : UChar
+0x090 ProxyData         : Ptr32 _SECURITY_TOKEN_PROXY_DATA
+0x094 AuditData         : Ptr32 _SECURITY_TOKEN_AUDIT_DATA
+0x098 OriginatingLogonSession : _LUID
+0x0a0 VariablePart      : Uint4B
```

The UserAndGroups offset (0x068) is a pointer to an SID_AND_ATTRIBUTES structure (again XP SP2):

```
dt_SID_AND_ATTRIBUTES
+0x000 Sid      : Ptr32 Void
+0x004 Attributes : Uint4B
```

So by tracing the chain of structures at the correct offsets, the SID (Security Identifier) of the owner of the Process can be determined. It should be pointed out that the association with the username can not readily be determined from memory only. The SID will have to be compared with System registry or domain specific information (correlated with information from the non-volatile stores).

The pointer to the Access Token is a virtual address and must be converted to a physical address, this typically involves breaking the virtual address into it's parts: 10 bits, Page Directory Index, 10 bits Page Table Index, 12 bits Page Offset. (PDI, PTI and POFF respectively)

The Page Directory Base(PDB) (another EPROCESS value) and the Page Directory Index (PDI) multiplied by the page size (4) will yield the ID of the a Page Table Entry (PTE1). The first 20 bits of PTE1 are concatenated with the Page Table Index multiplied by 4 to yield an ID to another Page Table Entry (PTE2). The first 20 bits of this Page Table Entry can be concatenated with the Page Offset to yield the physical address.

Example:

AccessToken = e3892033 (both obtained from EPROCESS)
 PDB = 06221000

(derived from the parts of AccessToken)

PDI = AccessToken / 0x400000 = 38e

PTI = AccessToken % 0x400000 / 0x1000 = 92

POFF = AccessToken % 0x1000 = 33

PTE1_ID = PDB + PDI * 4 = 6221e38

(value of PTE1_ID looked up in memory image)

PTE1 = 00831963

PTE2_ID = PTE1 - (PTE1 % 0x1000) + PTI * 4 = 831248

(value of PTE2_ID looked up in memory image)

PTE2 = 12319963

Physical Addr = PTE2 - (PTE2 % 0x1000) + POFF = 12319033

Unfortunately, even though that the physical address of the Access Token is now known, obtaining the SID is still not a simple offset. Some variable length data fields are located in the Access Token “above” the SID. Results from this project show that the SID begins somewhere between 0x198 and 0x1D5 from this point, but this is from example, it is not derived or calculated. Once the SID is located it must be decoded. The format, like FileTime, is not straightforward.

A SID is typically seen to users and administrators as :

S-01-5-21-791032918-1291200457-768897840-500

However the same SID in it’s binary form appears as:

0105000000000005150000005634262FC927F64C3073D42DF4010000

Table 5 : SID Encoding shows the meaning of the different parts of the SID and the types of encoding used.

| Table 5 : SID Encoding | | | |
|------------------------|------------------|----------------------|------------------------|
| SID | Encoded Part | Description | Encoding |
| S-01 | 0x01 | SID Revision | 1 byte |
| | 0x05 | Number of dashes - 2 | 1 byte |
| 5 | 0x00000000000005 | NT Authority | 6 bytes, Big Endian |
| 21 | 0x000000015 | NT Non Unique | 4 bytes, Little Endian |
| 791032918 | 0x2f263456 | Domain Issuer Part 1 | 4 bytes, Little Endian |
| 1291200457 | 0x4cf627c9 | Domain Issuer Part 2 | 4 bytes, Little Endian |
| 768897840 | 0x2dd47330 | Domain Issuer Part 3 | 4 bytes, Little Endian |
| 500 | 0x000001f4 | User / Computer ID | 4 bytes, Little Endian |

There are some well known SIDs and parts of SIDs:

| | |
|------------|---|
| S-1-1-0... | Everyone |
| S-1-2-0... | Locally Logged on |
| S-1-3-0... | Creator Owner ID |
| S-1-3-1... | Creator Group ID |
| ...500 | Administrator Account |
| ...501 | Guest Account |
| ...1000 | User Account |
| | (1000 and higher...1005 would indicated the 6 th user) |

Appendix F: Offset Deltas by service pack.

This appendix is provided to enable a more reader friendly format of different offsets of windows kernel structures. The offsets shown here (numeric values are hexadecimal) are also available in the proof of concept source code listing in another appendix.

| Table 6 : Windows Data Structure Offsets | | | | |
|--|------|-----|---------|------|
| | 2000 | XP | XP SP 2 | 2003 |
| EP_PageDirBase | 18 | 18 | 18 | 18 |
| EP_processors | 34 | 34 | 34 | 34 |
| EP_T_Forward | 50 | 50 | 50 | 50 |
| EP_T_Back | 54 | 54 | 54 | 54 |
| EP_priority | 62 | 62 | 62 | 62 |
| EP_T_Quantum | 63 | 63 | 6f | 63 |
| EP_T_Qant_dis | 69 | 69 | 69 | 69 |
| EP_exitStatus | 6c | 24c | 1d0 | 24c |
| EP_createTime | 88 | 70 | 70 | 70 |
| EP_exitTime | 90 | 78 | 78 | 78 |
| EP_PID(client Unique) | 9c | 84 | 84 | 84 |
| EP_WorkSetSize | e4 | 20c | 20c | 214 |
| EP_WorkSetMin | e8 | 210 | 210 | 1f8 |
| EP_WorkSetMax | ec | 214 | 214 | 1fc |
| EP_AccessToken | 12c | 0c8 | 0c8 | 0c8 |
| EP_PPID | 1c8 | 14c | 14c | 128 |
| EP_name | 1fc | 174 | 174 | 154 |
| EP_size | 290 | 258 | 260 | 278 |
| TH_size | 248 | 258 | 258 | 260 |
| TH_createTime | 1b0 | 1c0 | 1c0 | 1c8 |
| TH_exitTime | 1b8 | 1c8 | 1c8 | 1d0 |
| TH_exitStatus | 1c0 | 1d0 | 1d0 | 1d8 |
| TH_PID (client unique) | 1e0 | 1ec | 1ec | 1f4 |
| TH_TID (client unique) | 1e4 | 1f0 | 1f0 | 1f8 |
| TH_isTerminated | 224 | 248 | 248 | 250 |
| TH_startAddr | 230 | 224 | 224 | 22c |
| | | | | |

Appendix G: Proof of Concept Source Code Listing

The following PERL source will parse the contents of a memory image. On a 1.5 Mhz Pentium 3m with 1 GB of RAM it currently takes about 30 minutes to parse a 1GB image and 60 minutes to parse a 4 GB image. On a 2.66 Mhz Pentium 4 Xeon with 512MB Ram, it takes about 20 minutes to parse a 1GB image and 60 minutes to parse a 4GB image.

```
#!/usr/bin/perl
use strict;
use Getopt::Std;
use POSIX qw(strftime);
use Digest::MD5 qw(md5);

#variables that pertain more to this script than to the concept..
my $dump_header = 4096; # assuming size of dump file header - skipping
this is required to decode virtual addresses
my $isdump = 1;          # boolean if it is a dmp file - assume not
my $header_size = 0;     # most files will have no header
my $doprocess = 1;       # boolean to process processes
my $dothread = 0;        # boolean to process threads
my $simple = 0;           # simple mode (task manager view)
my $showoffsets = 1;     # show offsets - more of a CS view than a CJ
view
my $isunique = 1;        # use md5 to check for identical process structures
in memory, not really sure why you wouldn't want this
my $debug = 0;           #basically an output adjuster - mainly for
development
my $pThreshold = 0;      #adjust a variance for how many tests can fail
for process checking
my $tThreshold = 0;      #..and for threads
my $version = "procloc 0.6";
my $OS = "2K";           #Should be able to autodect this - or at least
command line option
my $output = 12;         #basically bitmask for output, 1 = name, 2 = pid, 3 =
name+pid, 4 = priority, 5 = name+priority, etc

#"super globals"  these variables are apparently the same for all
versions encountered today.
# but of course they may have to be broken down by OS at a later date.
my $kernelBound = 0x80000000; # all windows kernel virtual addresses
are above 80000000 (except /3G systems)
my $pagesize = 4096;     # size of a typical intel page
```

```

my $memSegBound = 8; # from MS driver memory article, almost a speed
factor increase
        #Six tips for efficient memory usage
        #http://www.microsoft.com/whdc/driver/perform/mem-
alloc.mspx

#my $SIZEOF_PROC = 0x290; #actually i think these are different by
OS...
#my $SIZEOF_THRD = 0x248;

my $currentpos = 0;
my $count = 0;
#my $test;
my %uniqueprocs;
my %uniquethreads;

#The DISPATCH HEADER is the key to locating the different structures in
memory. This header is
#used by processes (the original goal of this project) as well as other
structures. The
#signature of these types change by OS version. These multiple hashes
are switched between...
#...it could be implemented as a hash of a hash, but that usually ends
up losing people...
# I changed it to one hash per OS...for DH, Proc, and Thread
info...maybe a multi hash is the way...
#DH header type, from Windows Internals, 2005
#0 notification
#1 synchronizaiton
#2 mutant
#3 process
#4 queue
#5 semaphore
#6 thread
#8 notification timer
#9 sync timer

#This probably could be done with a single data structure, but hey...
my %win2K = (
#     DH_notification => "\x00.",
#     DH_notificationsize => "\x00.",
#     DH_sync => "\x01.",
#     DH_syncsize => "\x04.",
#     DH_mutant => "\x02.",
#     DH_mutantsize => "\x00.",
#     DH_process => "\x03.",
#     DH_processsize => "\x1b.",
#     DH_queue => "\x04.",
#     DH_queuesize => "\x00.",
#     DH_semaphore => "\x05.",
#     DH_semaphoresize => "\x05.",
#     DH_thread => "\x06.",
#     DH_threadsize => "\x6c.",
#     DH_notificationtimer => "\x08.",

```

```

        DH_notificationtimersize => "\x0a.",
#       DH_synctimer => "\x.",
#       DH_synctimersize => "\x.",
        EP_processors => 0x034,
        EP_priority => 0x062,
        EP_TH_Quantum => 0x63,
        EP_TH_Quantum_Disable => 0x69,
        EP_size => 0x290,
        EP_pageDirectoryBase => 0x018,
        EP_tListFlink => 0x050,
        EP_tListBlink => 0x054,
        EP_exitStatus => 0x06c,
        EP_createTime => 0x088,
        EP_exitTime => 0x090,
        EP_PID => 0x09c,
        EP_AccessToken => 0x12c,
        EP_AccessTokenSID => 0x188,
        EP_PPID => 0x1c8,
        EP_name => 0x1fc,
        TH_size => 0x248,
        TH_createTime => 0x1b0,
        TH_exitTime => 0x1b8,
        TH_exitStatus => 0x1c0,
        TH_PID => 0x1e0,
        TH_TID => 0x1e4,
        TH_isTerminated => 0x224,
        TH_tProcess => 0x22c,
        TH_startAddr => 0x230,
        EP_Win32P => 0x214,
        EP_WorkingSetSize => 0x0e4,
        EP_WorkingSetMin => 0x0e8,
        EP_WorkingSetMax => 0x0ec,
        EP_CommitChargeLimit => 0x268,
        EP_CommitChargePeak => 0x26c,
    );
my %winXP = (
#       DH_notification => "\x00.",
#       DH_notificationsize => "\x00.",
#       DH_sync => "\x01.",
#       DH_syncsize => "\x04.",
#       DH_mutant => "\x02.",
#       DH_mutantsize => "\x00.",
        DH_process => "\x03.",
        DH_processsize => "\x1b.",
#       DH_queue => "\x04.",
#       DH_queuesize => "\x00.",
        DH_semaphore => "\x05.",
        DH_semaphoresize => "\x05.",
        DH_thread => "\x06.",
        DH_threadsize => "\x70.",
        DH_notificationtimer => "\x08.",
        DH_notificationtimersize => "\x0a.",
#       DH_synctimer => "\x.",
#       DH_synctimersize => "\x.",

```

```

    EP_size => 0x258,
    EP_pageDirectoryBase => 0x018,
    EP_tListFlink => 0x050,
    EP_tListBlink => 0x054,
    EP_exitStatus => 0x24c,
    EP_createTime => 0x070,
    EP_exitTime => 0x078,
    EP_PID => 0x084,
    EP_PPID => 0x14c,
    EP_name => 0x174,
    TH_size => 0x258,
    TH_createTime => 0x1c0,
    TH_exitTime => 0x1c8,
    TH_exitStatus => 0x1d0,
    TH_PID => 0x1ec,
    TH_TID => 0x1f0,
    TH_isTerminated => 0x248,
    TH_tProcess => 0x220,
    TH_startAddr => 0x224,
);
my %winXP2 = (
#   DH_notification => "\x00.",
#   DH_notificationsize => "\x00.",
#   DH_sync => "\x01.",
#   DH_syncsize => "\x04.",
#   DH_mutant => "\x02.",
#   DH_mutantsize => "\x00.",
#   DH_process => "\x03.",
#   DH_processsize => "\x1b.",
#   DH_queue => "\x04.",
#   DH_queuesize => "\x00.",
#   DH_semaphore => "\x05.",
#   DH_semaphoresize => "\x05.",
#   DH_thread => "\x06.",
#   DH_threadsize => "\x70.",
#   DH_notificationtimer => "\x08.",
#   DH_notificationtimersize => "\x0a.",
#   DH_synctimer => "\x.",
#   DH_synctimersize => "\x.",
    EP_processors => 0x034,
    EP_priority => 0x062,
    EP_TH_Quantum => 0x6f,
    EP_TH_Quantum_Disable => 0x69,
    EP_size => 0x260,
    EP_pageDirectoryBase => 0x018,
    EP_tListFlink => 0x050,
    EP_tListBlink => 0x054,
    EP_AccessToken => 0x0c8,
    EP_createTime => 0x070,
    EP_exitTime => 0x078,
    EP_PID => 0x084,
    EP_PPID => 0x14c,
    EP_name => 0x174,
    EP_exitStatus => 0x24c,

```

```

    TH_size => 0x258,
    TH_createTime => 0x1c0,
    TH_exitTime => 0x1c8,
    TH_exitStatus => 0x1d0,
    TH_PID => 0x1ec,
    TH_TID => 0x1f0,
    TH_isTerminated => 0x248,
    TH_tProcess => 0x220,
    TH_startAddr => 0x224,
    EP_Win32P => 0x130,
    EP_CommitChargeLimit => 0x08a,
    EP_CommitChargePeak => 0x0ac,
    EP_WorkingSetSize => 0x20c,
    EP_WorkingSetMin => 0x210,
    EP_WorkingSetMax => 0x214,
);
my %win2003 = (
#   DH_notification => "\x00.",
#   DH_notificationsize => "\x00.",
#   DH_sync => "\x01.",
#   DH_syncsize => "\x04.",
#   DH_mutant => "\x02.",
#   DH_mutantsize => "\x00.",
#   DH_process => "\x03.",
#   DH_processsize => "\x1b.",
#   DH_queue => "\x04.",
#   DH_queuesize => "\x00.",
#   DH_semaphore => "\x05.",
#   DH_semaphoresize => "\x05.",
#   DH_thread => "\x06.",
#   DH_threadsize => "\x72.",
#   DH_notificationtimer => "\x08.",
#   DH_notificationtimersize => "\x0a.",
#   DH_synctimer => "\x.",
#   DH_synctimersize => "\x.",
    EP_size => 0x278,
    EP_pageDirectoryBase => 0x018,
    EP_tListFlink => 0x050,
    EP_tListBlink => 0x054,
    EP_exitStatus => 0x24c,
    EP_createTime => 0x070,
    EP_exitTime => 0x078,
    EP_PID => 0x084,
    EP_PPID => 0x128,
    EP_name => 0x154,
    TH_size => 0x260,
    TH_createTime => 0x1c8,
    TH_exitTime => 0x1d0,
    TH_exitStatus => 0x1d8,
    TH_PID => 0x1f4,
    TH_isTerminated => 0x250,
    TH_tProcess => 0x228,
    TH_startAddr => 0x22c,
);

```

```
# command line options (see usage() for more info)
my %opts;
my $opt_string = 'svahtTpPuVd:o:O: ';
getopts( "$opt_string", \%opts) or usage();

#i realize that a lot of this is somewhat redundant, but i haven't
decided on the default script operation yet
if($opts{t}){ $dothread = 1; }
if($opts{T}){ $dothread = 0; }
if($opts{p}){ $doprocess = 1; }
if($opts{P}){ $doprocess = 0; }
if($opts{d}){ $debug = $opts{d}; print "User specified debug set to
$debug\n"; }
if($opts{u}){ print "$version\n"; usage(); exit(0); }
if($opts{h}){ print "$version\n"; usage(); exit(0); }
if($opts{V}){ print "$version\n"; }
if($opts{o}){ $OS = $opts{o}; print "User specified OS set to $OS\n";
}
if($opts{s}){
    $simple = 1;
    $dothread = 0;
    $doprocess = 1;
    $showoffsets = 0;
    $output = 7;
}
if($opts{v}){
    print "verbose!";
    $dothread = 1;
    $doprocess = 1;
    $debug = 10;
    $showoffsets = 1;
    $output = 32767;
}
if($opts{a}){
    print "do all\n";
    $dothread = 1;
    $doprocess = 1;
    $debug = 10;
    $showoffsets = 1;
    $output = 32767;
}
if($opts{O}){
    $output = $opts{O};
    print "user output set $output = ";
    $output = unpack("B*", pack("N", $output));
    print " $output\n";
}

my %OSoff;    #placeholder hash for the correct offsets for a particular
OS

if($OS eq "2K"){
```



```

        %OSoff = %win2K;
    }elseif($OS eq "XP"){
        %OSoff = %winXP;
    }elseif($OS eq "XP2"){
        %OSoff = %winXP2;
    }elseif($OS eq "2003"){
        %OSoff = %win2003;
    }else{
        print "unknown OS specified.\n";
        usage();
        exit(1);
    }

#if it is a dmp file, skip the header...not even needed really, unless
the header is not
#a multiple of the segBound
if($isdump){
    $header_size = $dump_header;
}

my $DH_SYNCRONIZATION_EVENT = "$OSoff{DH_sync}$OSoff{DH_syncsize}";
my $DH_PROCESS = "$OSoff{DH_process}$OSoff{DH_processsize}";
my $DH_SEMAPHORE = "$OSoff{DH_semaphore}$OSoff{DH_semaphoresize}";
my $DH_THREAD = "$OSoff{DH_thread}$OSoff{DH_threadsize}";
my $DH_NOTIFICATION_TIMER =
"$OSoff{DH_notificationtimer}$OSoff{DH_notificationtimersize}";

my $SIZEOF_PROC = $OSoff{EP_size};
my $SIZEOF_THRD = $OSoff{TH_size};

# get memory dump to parse
my $INFILE = shift;

if(!($INFILE)){ usage();}

### ready.....GO

open(INFILE, "<", $INFILE) or die "$0: unable to open $INFILE.";
binmode(INFILE);

if($isdump){
    sysseek(INFILE, $dump_header, 0);    #skip header, if present
}

header();

#work through memory memSegBound at a time
my $sentinel;
my $test;
my $break = 0;
while (($sentinel = sysread(INFILE, $test, $memSegBound)) && $break ==
0) {

```

```

$currentpos = sysseek(INFILE, 0, 1);
my $lpos = $currentpos - $pagesize;

#currently only processes and threads are tested, any DH type
could be implemented here though
if (substr($test, 0, 4) =~ /$DH_PROCESS/) {
    if ($doprocess){
        &ProcessTest ;
    }
} elsif (substr($test, 0, 4) =~ /$DH_THREAD/) {
    if ($dothread){
        &ThreadTest ;
    }
}

if($sentinel != $memSegBound){
    if($debug > 1){
        print "terminating condition found, sysread()
returned $sentinel, not $memSegBound";
    }
    $break=1; #annoying workaround due to "use strict" and
wanting the debug message...should be a better way
}
}

close(INFILE);
my $time = time() - $^T;
my $sec = $time % 60;
my $min = ($time - $sec) / 60;
print "\n Found $count structures in $min m $sec s\n";

#the FILETIME format store 100ns increments since Jan 1, 1601 in 64
bits
#unix stores 1 second interval since Jan 1, 1970
#to keep the program uniform one must be converted to the other

# Filetime conversions
# FFFFFFF00 00000000      = under 1.5 seconds
# 00000001 00000000      = under 1.5 seconds
# 00000010 00000000      = about 26 seconds
# 00000000 01000000      = about 7:09
# 00000000 10000000      = about 1:51:31
# 00000000 00010000      = about 1 day 6:32:31
# 00000000 00100000      = about 21 days 8:40:18
# 00000000 00000100      = about 11 months 22 days 18:44:57
# 00000000 00001000      = about 14 years 3 months 10 months 11:59:22
# 00000000 00000001      = about 228 years 5 months 5 days 23:50:03
# 00000000 00000010      = about 6353 years 6 months 18 days 21:21:00

# Using this understanding, of the endian-ness and order of the bytes
the FILETIME representation of the Unix Epoch (Jan 1, 1970) is 000040d5
deb19d01.

```

```

# Some of this understanding came from these websites
# http://aspn.activestate.com/ASPN/Mail/Message/perl-win32-
admin/1981214
#
http://www.koders.com/c/fidAB384423820A5D2FBF749480D6615D03E554271C.asp
x

# Convert win32 FILETIME to unix timestamp
sub Win2Unix4() {
    my $Lval = shift;
    my $Hval = shift;
    my $Time = 0;
    my $Shift = 11644473600; # win / unix epoch shift value obtained using
    a FILETIME shift of the unix Epoch

    if(($Lval == 0) and ($Hval == 0)){
        return $Time;
    }else{
        $Time = int(($Hval * 2**32 / 100000000) + ($Lval / 100000000));
        $Time -= $Shift;
    }

    if ($Time < 0){
        $Time = 0;
    }

    return $Time;
}

#String Format timestamp to human readable form (descending
significance)
sub sPrintTime() {

    my $Time = shift;
    my $Result;

    my ($sec,$min,$hour,$mday,$mon,$year,$yday,$wday) =
    gmtime($Time);

    if ($Time == 0) {
        $Result = '';
    } else {
        $Result = strftime("%Y.%m.%d %H:%M:%S", $sec, $min, $hour,
$mday, $mon, $year, -1, -1, -1);
    }

    return $Result;
}

#print out the first line - the column headers
sub header() {
    if($output) { printf(" %-3s",
"Cnt"); }
    if($output % 10 == 1) { printf(" %-16s",
"Name"); }
}

```

```

        if($output / 10 % 10 == 1)          { printf(" %-3s",
"Typ"); }
        if($output / 100 % 10 == 1)         { printf(" %-10s", "PID
( TID)"); }
        if($output / 1000 % 10 == 1)        { printf(" %-4s",
"Pri"); }
        if($output / 10000 % 10 == 1)       { printf(" %-15s",
"WorkSet"); }
        if($output / 100000 % 10 == 1)      { printf(" %-
19s", "Created"); }
        if($output / 1000000 % 10 == 1)     { printf(" %-
19s", "Terminated"); }
        if($output / 10000000 % 10 == 1)    { printf(" %-4s",
"Proc"); }
        if($output / 100000000 % 10 == 1)   { printf(" %-4s",
"Quan"); }
        if($output / 1000000000 % 10 == 1)  { printf(" %-4s",
"Quad"); }
        if($output / 10000000000 % 10 == 1) { printf(" %-10s",
"Offset"); }
        if($output / 100000000000 % 10 == 1) { printf(" %-
10s", "PDB"); }
        if($output / 1000000000000 % 10 == 1) { printf(" %-
10s", "AToken"); }
        return;
    }

```

#Test a potential processes

```

sub ProcessTest {
    if($debug > 2){ print("Found process candidate at
$currentpos.\n"); }
    my $potentialp;
    sysread(INFILE, $potentialp, $SIZEOF_PROC-$memSegBound,
$memSegBound);

    # unpack (size , what)  L = unsigned long, l = signed long, a* =
    ascii string, c = char, C = uchar

    my $PageDirectoryBase = unpack('L', substr($potentialp,
$OSoff{EP_pageDirectoryBase}, 4));
    my $ThreadListHeadFlink = unpack('L', substr($potentialp,
$OSoff{EP_tListFlink}, 4));
    my $ThreadListHeadBlink = unpack('L', substr($potentialp,
$OSoff{EP_tListBlink}, 4));
    my $ExitStatus = unpack('L', substr($potentialp,
$OSoff{EP_exitStatus}, 4));
    my $CreateTime2 = unpack('h*', substr($potentialp,
$OSoff{EP_createTime}, 8));
    my $CreateTimeLo = unpack('L', substr($potentialp,
$OSoff{EP_createTime}, 4));
    my $CreateTimeHi = unpack('l', substr($potentialp,
$OSoff{EP_createTime}+4, 4));

```

```

        my $ExitTimeLo          = unpack('L', substr($potentialp,
$OSoff{EP_exitTime}, 4));
        my $ExitTimeHi          = unpack('l', substr($potentialp,
$OSoff{EP_exitTime}+4, 4));
        my $PID                 = unpack('L', substr($potentialp,
$OSoff{EP_PID}, 4));
        my $Priority             = unpack('c', substr($potentialp,
$OSoff{EP_priority}, 16));
        my $Quantum             = unpack('c', substr($potentialp,
$OSoff{EP_TH_Quantum}, 16));
        my $QuantumD            = unpack('C', substr($potentialp,
$OSoff{EP_TH_Quantum_Disable}, 16));
        my $AcrProcs            = unpack('L', substr($potentialp,
$OSoff{EP_processors}, 4));
        my $Win32P              = unpack('L', substr($potentialp,
$OSoff{EP_Win32P}, 4));
        my $CommitLim           = unpack('L', substr($potentialp,
$OSoff{EP_CommitChargeLimit}, 4));
        my $CommitPeak          = unpack('L', substr($potentialp,
$OSoff{EP_CommitChargePeak}, 4));
        my $WorkingSetSize      = unpack('L', substr($potentialp,
$OSoff{EP_WorkingSetSize}, 4));
        my $WorkingSetMin       = unpack('L', substr($potentialp,
$OSoff{EP_WorkingSetMin}, 4));
        my $WorkingSetMax       = unpack('L', substr($potentialp,
$OSoff{EP_WorkingSetMax}, 4));
        my $AccessToken         = unpack('L', substr($potentialp,
$OSoff{EP_AccessToken}, 4));

        #these are all essentially bit masks
        my $ATPDI = $AccessToken / 0x400000;
        my $ATPTI = $AccessToken % 0x400000 / 0x1000;
        my $ATOFF = $AccessToken % 0x1000;

        my $PTE1_ID = $PageDirectoryBase + ($ATPDI * 0x4);
        if($debug > 1){
            printf("PTE ID 0x%0.8x\n", $PTE1_ID);
        }

        my $temp;
        #ok, this may get hairy...we're going to re-use the filehandle
and jump around decoding the virtual memory...
        #this isn't quite working yet...
        sysseek(INFILE, $PTE1_ID, 0);
        sysread(INFILE, $temp, 4);
        my $PTE1 = unpack('V', $temp);
        if($debug > 1){ printf("PTE Val 0x%0.8x %s\n", $PTE1);}

        $PTE1 = $PTE1 - $PTE1 % 0x1000;
        my $PTE2_ID = $PTE1 + ($ATPTI * 0x4)-1;    #???
        if($debug > 1){ printf("PTE2 ID 0x%0.8x %0.8x %0.8x \n",
$PTE2_ID, $PTE1, $ATPTI * 4);}

        sysseek(INFILE, $PTE2_ID, 0);

```

```

sysread(INFILE, $temp, 4);
my $PTE2 = unpack('V', $temp);
    if($debug > 1){ printf("PTE2 Val 0x%0.8x %s\n", $PTE2);}
    $PTE2 = $PTE2 - $PTE2 % 0x1000;

my $PageBaseAddress = $PTE2 + $ATOFF;
    if($debug > 1){ printf("Page Base Addr %x \n",
$PageBaseAddress);}

my $useroffset = $PageBaseAddress + 0x198;

    if($debug > 1){ printf ("uo %0.8x\n", $useroffset);}
sysseek(INFILE, $useroffset, 0);
sysread(INFILE, $temp, 28);

my $uSID = SIDbin2ascii($temp);

#done with the virutal memory part
#put the current spot in the file back so we can continue on with
the search
sysseek(INFILE, $currentpos+$SIZEOF_PROC-$memSegBound, 0);

my $PPID          = unpack('L', substr($potentialp,
$OSoff{EP_PPID}, 4));
my $ImageFileName = unpack('a*', substr($potentialp,
$OSoff{EP_name}, 16));

my $CreateTime    = &Win2Unix4($CreateTimeLo, $CreateTimeHi);
my $ExitTime= &Win2Unix4($ExitTimeLo, $ExitTimeHi);

my $ptestcount=0;

#except for IDLE, a process must have a priority
if ($Priority ==0 && $PID !=0){
    $ptestcount++;
    if($debug > 2){ print("Test failed: Process (other than
IDLE) has a priority of 0 (or lower).\n");}
}

#windows is supposed to have only 32 process levels...
#0 - Idle (system level), 1-15 'variable level', and 16-31 'real-
time'
if ($Priority < 0 || $Priority > 31){
    $ptestcount++;
    if($debug > 2){ print("Test failed: Process is out of
Priority Level Range (0-31).\n");}
}

#page directory must exist
if ($PageDirectoryBase == 0) {
    $ptestcount++;

```

```

        if($debug > 2){ print("Test failed: PageDirectoryBase is
NULL.\n");}
    }

    # PDB has to start at a pageboundary.
    if ($PageDirectoryBase % $pagesize != 0) {
        $ptestcount++;
        if($debug > 2){ print("Test failed: PageDirectoryBase not
aligned on page boundary.\n");}
    }

    # all threads must be in the kernel virtual memory space
    if ($ThreadListHeadFlink < $kernelBound) {
        $ptestcount++;
        if($debug > 2){ print("Test failed: ThreadList Flink does
not point into kernel space.\n");}
    }
    if ($ThreadListHeadBlink < $kernelBound) {
        $ptestcount++;
        if($debug > 2){ print("Test failed: ThreadList Blink does
not point into kernel space.\n");}
    }

    #Quantum is typically 2 clock intervals for xp / 2000 client and
12 for server platforms
    #but for a variety of reasons it's actually stored at a multiple
of 3 (so 6 and 36 respectively).
    if ($Priority ==0 && $PID !=0){
        $ptestcount++;
        if($debug > 2){ print("Test failed: Process (other than
IDLE) has a priority of 0 (or lower).\n");}
    }

    #workinstsetmin  50 / 0  (system)

    #workingsetmax   345 / 450 (idle)

    #VADs

    #access-token

    #thread count - not implimented currently becuae this would
require caching all output until the end of the scan
        # and this machine cannot handle this...

    #check for minimum required procesess (sysinternals has written
about the minimal set)

    #Check for default set of processes for heuristic

    #check for maximum (i know there is a max in linux, there may be
in windows...)

    #Check for sync events

```

```

        if ($ptestcount > 0 + $pThreshold) {
            if($debug > 2){ print("Test Failed: Not a process, failed
at $ptestcount");}
            sysseek(INFILE, $currentpos+$memSegBound, 0);
            return;
        }

        if ($isunique) {
            #various reasons the same process might exist in multiple
locations
            #tracking occurrences of the same process might be
interesting, but not done at this point
            my $hash = md5($potentialp);
            $uniqueprocs{$hash}++;
            if ($uniqueprocs{$hash} > 1) {
                # print("Duplicate process detected
?X?X?!!!!!!!!!!!!!!!!!!!!!!!!!!!!11\n
#####. \n");
                if($debug > 2){ print("Duplicate process detected
. \n");}
                return;
            }
        }
        if($debug > 2){ print(" SUCCESS: Good proc found!.\n");}
#         if($simple == 1){
#             print "D\n";
#         }else{
            printf("\n%4d", ++$count);
            if($output % 10 == 1) { printf(" %16s",
$ImageFileName); }
            if($output / 10 % 10 == 1) { printf(" %3s", " P
"); }
            if($output / 100 % 10 == 1) { printf(" %4d ",
$PID); }
            if($output / 1000 % 10 == 1) { printf(" %4d",
$Priority); }
            if($output / 10000 % 10 == 1) { printf(" %15d",
$WorkingSetSize*($pagesize/1024)); }
            if($output / 100000 % 10 == 1) { printf(" %19s",
&$PrintTime($CreateTime)); }
            if($output / 1000000 % 10 == 1) { printf(" %19s",
&$PrintTime($ExitTime)); }
            if($output / 10000000 % 10 == 1) { printf(" %4s",
$AcrProcs); }
            if($output / 100000000 % 10 == 1) { printf(" %4d",
$Quantum); }
            if($output / 1000000000 % 10 == 1) { printf(" %4d",
$QuantumD); }
            if($output / 10000000000 % 10 == 1) { printf(" 0x%0.8x",
$currentpos); }
            if($output / 100000000000 % 10 == 1) { printf("
0x%0.8x", $PageDirectoryBase); }

```



```

        if($output / 1000000000000 % 10 == 1)      { printf("
0x%0.8x", $AccessToken);}

}

# test a potential thread
sub ThreadTest() {
    if($debug > 2){ print("Found thread candidate at $currentpos
\n");}
    my $potentialt;
    sysread(INFILE, $potentialt, $SIZEOF_THRD-$memSegBound,
$memSegBound);

    my $CreateTimeLo = unpack('L', substr($potentialt,
$OSoff{TH_createTime}, 4));
    my $CreateTimeHi = unpack('l', substr($potentialt,
$OSoff{TH_createTime}+4, 4));
    my $ExitTimeLo    = unpack('L', substr($potentialt,
$OSoff{TH_exitTime}, 4));
    my $ExitTimeHi    = unpack('l', substr($potentialt,
$OSoff{TH_exitTime}+4, 4));
    my $ExitStatus    = unpack('L', substr($potentialt,
$OSoff{TH_exitStatus}, 4));
    my $PID           = unpack('L', substr($potentialt,
$OSoff{TH_PID}, 4));
    my $TID           = unpack('L', substr($potentialt,
$OSoff{TH_PID}+4, 4));
    my $HasTerminated = unpack('L', substr($potentialt,
$OSoff{TH_isTerminated}, 4));
    my $ThreadsProcess = unpack('L', substr($potentialt,
$OSoff{TH_tProcess}, 4));
    my $StartAddress  = unpack('L', substr($potentialt,
$OSoff{TH_startAddr}, 4));
    my $Win32StartAddress = unpack('L', substr($potentialt,
$OSoff{TH_startAddr}+4, 4));
    my $ExitTime      = &Win2Unix4($ExitTimeLo, $ExitTimeHi);
    my $CreateTime    = &Win2Unix4($CreateTimeLo,
$CreateTimeHi);

    my $ttestcount = 0;

    if (($ThreadsProcess < $kernelBound) && ($PID != 0)) {
        $ttestcount++;
        if($debug > 2){ print("Test failed: ThreadsProcess not in
kernel space.\n");}
    }
    if (($StartAddress == 0) && ($PID != 0)) {
        $ttestcount++;
        if($debug > 2){ print("Test failed: StartAddress is
NULL.\n");}
    }

    #thread priority base / current (if altered from base)

```

```

# extra checks on structures
if (substr($potentialt, 0x0e8, 4) !~ /$DH_NOTIFICATION_TIMER/) {
    $ttestcount++;
    if($debug > 2){ print("Test failed: No NOTIFICATION_TIMER
at 0x0e8.\n");}
}
if (substr($potentialt, 0x190, 4) !~ /$DH_SEMAPHORE/) {
    $ttestcount++;
    if($debug > 2){ print("Test failed: No SEMAPHORE at
0x190.\n");}
}
if ((substr($potentialt, 0x1e8, 4) !~ /$DH_SEMAPHORE/) && ($PID
!= 0)) {
    $ttestcount++;
    if($debug > 2){ print("Test failed: No SEMAPHORE at
0x1e8.\n");}
}

#determine if this one was a thread
if ($ttestcount > 0 + $tThreshold) {
    if($debug > 2){ print(" FAILURE: bad thread,
skipping.\n");}
    #..if not, move to next test location and start the whole
process over
    sysseek(INFILE, $currentpos+$memSegBound, 0);
    return;
}

#similar to process situation
if ($isunique) {
    my $hash = md5($potentialt);
    $uniquethreads{$hash}++;
    if ($uniquethreads{$hash} > 1) {
        if($debug > 2){ print("Duplicate thread found.\n");}
        return;
    }
}

if($debug > 2){ print(" SUCCESS: Found good thread!!.\n");}
printf("\n%4d", ++$count);
if($output % 10 == 1) { printf(" %16s", " ");}
if($output / 10 % 10 == 1) { printf(" %3s", " T
"); }
if($output / 100 % 10 == 1) { printf(" %4d(%4d)",
$PID,$TID); }
if($output / 1000 % 10 == 1) { printf(" %4d", " ");}
if($output / 10000 % 10 == 1) { printf(" %15d", "
");}
if($output / 100000 % 10 == 1) { printf(" %19s",
&sPrintTime($CreateTime));}

```

```

        if($output / 1000000 % 10 == 1)          { printf(" %19s",
&sPrintTime($ExitTime));}
        if($output / 10000000 % 10 == 1)        { printf(" %4s", " ");}
        if($output / 100000000 % 10 == 1)       { printf(" %4d", " ");}
        if($output / 1000000000 % 10 == 1)      { printf(" %4d", " ");}
        if($output / 10000000000 % 10 == 1)     { printf(" 0x%0.8x",
$currentpos);}
        if($output / 100000000000 % 10 == 1)    { printf("
0x%0.8x", " ");}
        if($output / 1000000000000 % 10 == 1)   { printf("
0x%0.8x", " ");}
    }

```

#input is binary windows SID value

#output is human readable SID

#formula is defined at

<http://blogs.msdn.com/oldnewthing/archive/2004/03/15/89753.aspx>

#bytes format

```

#1    revision (S-1)
#1    number of dashes minus 2
#6    security big endian
#4    non unique little endian
#4    domain id little endian
#4    domain id little endian
#4    user/machine id little endian

```

#convert a binary SID to ASCII

```

sub SIDbin2ascii(){
    my $SID = shift;

    my $s1 = unpack('H*', substr($SID, 0, 1));
    my $s2 = unpack('H*', substr($SID, 1, 1));
    my $s3 = unpack('H*', substr($SID, 2, 6));
    my $s4 = unpack('V',  substr($SID, 8, 4));
    my $s5 = unpack('V',  substr($SID, 12, 4));
    my $s6 = unpack('V',  substr($SID, 16, 4));
    my $s7 = unpack('V',  substr($SID, 20, 4));
    my $s8 = unpack('V',  substr($SID, 24, 4));

    return sprintf ("S-%s-%x-%s-%s-%s-%s-%s\n", $s1, $s3, $s4, $s5,
$s6, $s7, $s8);
}

```

```

sub usage(){
    print<<END;

```

Procloc - Process Locator (\$version)

2006 Tim Vidas

Locates processes and similar structures in an image of windows memory obtained either by crash dump or by dd-style acquisition.

usage: ./procloc.pl [-hvdO] [-t|T] [-p|P] [-f file]

```
-h / u      : this (help) message
-t          / T    : show threads / do not
-p / P      : show processes / do not
-o          : os selection (2K, XP, XP2, 2003)
-d          : print debugging messages to stderr
-f file     : file containing usernames, one per line
-v          : verbose output
-s          : simple output
-a          : "all output"
-O          : set output options:
              1      name
              2      type (p=process, t=thread)
              4      pid (tid)
              8      priority
              16     Working Set
              32     created
              64     terminated
              128    procs
              256    quantum
              512    quantumdelta
              1024   offset
              2048   PDB
              4096   AccessToken
              i.e. 1029 would be Name, PID, and offset
              7     would be name, type, and PID
```

example: \$0 -v -d -f file

This program is free software; you can redistribute it and/or
 modify
 it under the terms of the GNU General Public License as published
 by
 the Free Software Foundation; version 2

This program is distributed in the hope that it will be useful,
 but WITHOUT ANY WARRANTY; without even the implied warranty of
 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 GNU General Public License for more details.

You should have received a copy of the GNU General Public License
 along with this program; if not, write to the Free Software
 Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-
 1301 USA

END

```
    exit(0);
}
```

Appendix H: Sample runs

Several different options are demonstrated to show the capabilities of the script and for paper print formatting reasons. Sample runs were created on an IBM Thinkpad R52 (1.5 Mhz Pentium 3m w/ 1 GB RAM). The type of OS the image was aquired from can be seen in the command line, all images are 512 MB in size.

```
C:\>procloc.pl -h
procloc 0.6
```

```
Procloc - Process Locator (procloc 0.6)
```

```
2006 Tim Vidas
```

```
Locates processes and similar structures in an image of windows
memory obtian
ed either by crash dump or by dd-style acquisition.
```

```
usage: ./procloc.pl [-hvdO] [-t|T] [-p|P] [-f file]
```

```
-h / u      : this (help) message
-t / T      : show threads / do not
-p / P      : show processes / do not
-o          : os selection (2K, XP, XP2, 2003)
-d          : print debugging messages to stderr
-f file     : file containing usernames, one per line
-v          : verbose output
-s          : simple output
-a          : "all output"
-O          : set output options:
              1      name
              2      type
              4      pid (tid)
              8      priority
              16     Working Set
              32     created
              64     terminated
              128    procs
              256    quantum
              512    quantumdelta
              1024   offset
              2048   PDB
              4096   AccessToken
              i.e. 1029 would be Name, PID, and offset
              7     would be name, type, and PID
```

```
example: C:\procloc.pl -v -d -f file
```


| | | | | | | | |
|----|--------------|---|----|---|------------|------------|------------|
| 19 | llssrv.exe | 0 | 36 | 0 | 0x01d1b028 | 0x068ba000 | 0xe1c7b250 |
| 20 | svchost.exe | 0 | 36 | 0 | 0x01d1f888 | 0x06c2e000 | 0xe1c75e10 |
| 21 | msdtc.exe | 0 | 36 | 0 | 0x01d35928 | 0x06626000 | 0xe1c6c750 |
| 22 | SPOOLSV.EXE | 0 | 36 | 0 | 0x01d3a848 | 0x064a1000 | 0xe1c675f0 |
| 23 | svchost.exe | 0 | 36 | 0 | 0x01d46508 | 0x06547000 | 0xe1c62270 |
| 24 | rundll32.exe | 0 | 36 | 0 | 0x01d5f708 | 0x1a87b000 | 0x00000000 |
| 25 | lsass.exe | 0 | 36 | 0 | 0x01d6cca8 | 0x05b2a000 | 0xe1c16030 |
| 26 | services.exe | 0 | 36 | 0 | 0x01d6e948 | 0x05a5a000 | 0xe1c13910 |
| 27 | winlogon.exe | 0 | 36 | 0 | 0x01d7eca8 | 0x05883000 | 0xe1b45590 |
| 28 | csrss.exe | 0 | 36 | 0 | 0x01e14028 | 0x04bfe000 | 0xe1aef3b0 |
| 29 | smss.exe | 0 | 36 | 0 | 0x01e31408 | 0x03b3a000 | 0xe1401e10 |
| 30 | System | 0 | 36 | 0 | 0x020449e8 | 0x00030000 | 0xe1001770 |

Found 30 structures in 6 m 42 s

C:\>procloc.pl -o XP2 -s f:\memory\xpsp2\xpsp2.dd

User specified OS set to XP2

| Cnt | Name | Typ | PID (TID) |
|-----|--------------|-----|------------|
| 1 | Idle | P | 0 |
| 2 | rundll32.exe | P | 1124 |
| 3 | helix.exe | P | 836 |
| 4 | SOUNDMAN.EXE | P | 1224 |
| 5 | msmsgs.exe | P | 1340 |
| 6 | alg.exe | P | 608 |
| 7 | nvsvc32.exe | P | 204 |
| 8 | regedit.exe | P | 1964 |
| 9 | spoolsv.exe | P | 1892 |
| 10 | svchost.exe | P | 1420 |
| 11 | svchost.exe | P | 1552 |
| 12 | svchost.exe | P | 1676 |
| 13 | svchost.exe | P | 1172 |
| 14 | svchost.exe | P | 1108 |
| 15 | lsass.exe | P | 920 |
| 16 | services.exe | P | 908 |
| 17 | wscntfy.exe | P | 736 |
| 18 | csrss.exe | P | 832 |
| 19 | explorer.exe | P | 1068 |
| 20 | wuauclt.exe | P | 1584 |
| 21 | winlogon.exe | P | 864 |
| 22 | smss.exe | P | 776 |
| 23 | System | P | 4 |

Found 23 structures in 6 m 58 s

C:\>procloc.pl -t -O 1031 f:\memory\2kssp3\MEMORY.DMP

user output set 1031 = 00000000000000000000000010000000111

| Cnt | Name | Typ | PID (TID) | Offset |
|-----|------|-----|-------------|------------|
| 1 | Idle | P | 0 | 0x0040d5e8 |
| 2 | | T | 0 (0) | 0x0040d878 |
| 3 | | T | 936 (524) | 0x01712ca8 |
| 4 | | T | 1764 (336) | 0x017e7da8 |
| 5 | | T | 284 (1384) | 0x0182d488 |
| 6 | | T | 924 (1572) | 0x0182e988 |
| 7 | | T | 596 (1520) | 0x0182fda8 |
| 8 | | T | 1684 (1768) | 0x01832028 |
| 9 | | T | 596 (1748) | 0x01833788 |

| | | | | |
|----|--------------|---|-------------|------------|
| 10 | | T | 1684 (1692) | 0x018359c8 |
| 11 | | T | 1684 (1688) | 0x01835da8 |
| 12 | | T | 596 (1704) | 0x01839da8 |
| 13 | | T | 596 (1652) | 0x0183a4e8 |
| 14 | | T | 244 (1640) | 0x0183bda8 |
| 15 | | T | 1516 (1616) | 0x0183e4e8 |
| 16 | | T | 244 (1628) | 0x0183f0e8 |
| 17 | | T | 244 (1348) | 0x01853148 |
| 18 | | T | 1516 (1636) | 0x018538c8 |
| 19 | | T | 232 (1632) | 0x01857848 |
| 20 | | T | 1516 (1624) | 0x0185b028 |
| 21 | | T | 1612 (1608) | 0x01864508 |
| 22 | SOUNDMAN.EXE | P | 1612 | 0x018647c8 |
| 23 | | T | 1516 (1604) | 0x01864a88 |
| 24 | | T | 1516 (1600) | 0x01864d08 |
| 25 | | T | 1516 (1596) | 0x01866428 |
| 26 | wuaucflt.exe | P | 284 | 0x0186a928 |
| 27 | | T | 1516 (1096) | 0x0186b728 |
| 28 | | T | 1516 (1568) | 0x0186d748 |
| 29 | | T | 1516 (1592) | 0x0186e9c8 |
| 30 | rundll32.exe | P | 1364 | 0x0186f1c8 |
| 31 | | T | 1516 (1564) | 0x018702e8 |
| 32 | | T | 184 (644) | 0x0187e028 |
| 33 | | T | 1672 (1760) | 0x01898028 |
| 34 | | T | 1672 (1756) | 0x01898b28 |
| 35 | | T | 1684 (1732) | 0x018a7028 |
| 36 | | T | 1684 (1696) | 0x018a8da8 |
| 37 | | T | 1516 (1556) | 0x018aa028 |
| 38 | | T | 596 (1664) | 0x018ab028 |
| 39 | | T | 284 (1392) | 0x018b0988 |
| 40 | | T | 1684 (1680) | 0x018b4028 |
| 41 | | T | 1516 (480) | 0x018b5028 |
| 42 | | T | 496 (1588) | 0x018b8028 |
| 43 | | T | 992 (1668) | 0x018b83a8 |
| 44 | | T | 948 (280) | 0x018b8628 |
| 45 | | T | 436 (880) | 0x018b88a8 |
| 46 | | T | 704 (372) | 0x018b8da8 |
| 47 | dd.exe | P | 1464 | 0x018ba028 |
| 48 | | T | 1764 (1776) | 0x018cfda8 |
| 49 | helix.exe | P | 1764 | 0x018d0028 |
| 50 | | T | 8 (1228) | 0x018d1aa8 |

<trimmed for brevity>

| | | | | |
|-----|--------------|---|------------|------------|
| 351 | SERVICES.EXE | P | 232 | 0x01acfac8 |
| 352 | | T | 180 (220) | 0x01ad0808 |
| 353 | | T | 184 (208) | 0x01ad2028 |
| 354 | | T | 184 (212) | 0x01ad2a48 |
| 355 | | T | 184 (204) | 0x01adc288 |
| 356 | | T | 180 (148) | 0x01adf8e8 |
| 357 | WINLOGON.EXE | P | 180 | 0x01adfb68 |
| 358 | | T | 184 (200) | 0x01ae0308 |
| 359 | | T | 184 (196) | 0x01ae0608 |
| 360 | | T | 184 (192) | 0x01ae0a28 |

```

-361          T      184 ( 188) 0x01ae22e8
362  CSRSS.EXE  P      184      0x01b960c8
363          T      160 ( 176) 0x01b96548
364          T      160 ( 172) 0x01b96888
365          T      160 ( 112) 0x01b96b28
366  SMSS.EXE   P      160      0x01bb3328
367          T        8 ( 152) 0x01bb36e8
368          T        8 ( 132) 0x01bb4028
369          T        8 ( 144) 0x01bb4b28
370          T        8 ( 140) 0x01bb4da8
371          T        8 ( 136) 0x01bb7088
372          T     160 ( 156) 0x01bba028
373          T     160 ( 168) 0x01bba9c8
374          T     160 ( 164) 0x01bbac68
375          T        8 ( 128) 0x01c0b1e8
376          T        8 ( 124) 0x01c180e8
377          T        8 ( 120) 0x01c18368
378          T        8 ( 108) 0x01fe1408
379          T        8 ( 104) 0x01fe7588
380          T        8 ( 100) 0x02035028
381          T        8 (  92) 0x0203b028
382          T        8 (  76) 0x0203f028
383          T        8 (  88) 0x0203f328
384          T        8 (  84) 0x0203fb28
385          T        8 (  80) 0x0203fda8
386          T        8 (  72) 0x020402e8
387          T        8 (  60) 0x02041028
388          T        8 (  68) 0x02041b28
389          T        8 (  64) 0x02041da8
390          T        8 (  36) 0x02042028
391          T        8 (  56) 0x020423a8
392          T        8 (  52) 0x02042628
393          T        8 (  48) 0x020428a8
394          T        8 (  44) 0x02042b28
395          T        8 (  40) 0x02042da8
396          T        8 (  12) 0x02043028
397          T        8 (  32) 0x020433a8
398          T        8 (  28) 0x02043628
399          T        8 (  24) 0x020438a8
400          T        8 (  20) 0x02043b28
401          T        8 (  16) 0x02043da8
402          T        8 (   4) 0x02044768
403  System     P        8      0x020449e8
404          T        8 (  96) 0x020596e8
Found 404 structures in 6 m 26 s

```

C:\>

Appendix I: Linux Acquisition

Kcore exists in the /proc filesystem, it is essentially a virtual view of physical RAM.

Kcore might be the preferred way to image memory in Linux because shows memory in a structured way: ELF format. Supposedly kcore is only 4kb larger than Physical RAM, experiments done as part of this project show that there may actually be larger difference in file size. Different distributions implement kcore differently and Kernel Development lists have even long considered removing kcore altogether (/proc/kcore May Be Going Away, 2003), so it's long term reliability may be questionable.

You can enumerate the file type of kcore with the file command, and creating an image of RAM is as simple as using the dd command:

```
$>file /proc/kcore
/proc/kcore: ELF 32-bit LSB core file Intel 80386, version 1 (SYSV),
SVR4-style, SVR4-style, from 'vmlinux'
$>ddcidd < /proc/kcore | <something to receive the memory – netcat?>
```

Redhat (and possibly other distributions) does not support reading kcore. To re-enable this feature, you have to uncomment the lines that limits its usage – basically restoring the functionality that exists in the non-Redhat-modified Linux kernel.

Savekore is available in unix and variants like Solaris, but not typically in Linux. To approximate some of the functionality presented earlier for Windows, there are several additions or modifications that can be performed to Linux systems. There is an Open

Source (sourceforge) project called Linux Kernel Crash Dump (LKCD), that requires kernel patching (pre-incident). Mission Critical Linux has a patch called MCore. Redhat Advanced Server 2.1+ (Enterprise Level - pay product) offers some tools like Netdump and Crash. Rational and implementation notes are available online (Johnson, 2002).

/dev/mem and /dev/kmem may very well be implemented in a more standard way, but are also more “dangerous”³³ to use as they are essentially devices. Similar to kcore, Redhat (and possibly others) restricts access, basically only allowing access to the first 1 MB.

Analysis

The methodology is similar to the proof on concept for Windows RAM dumps, with Linux the EPROCESS similar structure is task_struct, task_structs are doubly linked, pages are the same: 4096 bytes, and there is a kernel / user bound (though the bound is 1GB / 3GB, so similar to the /3GB Windows boot switch so the boundary would be 0xc0000000 instead of 0x80000000).

Gdb is an Open Source debugger provided with many distributions of Linux. Symbols are available in Linux and, just as in windows, can aide in debugging binaries by helping associate function names, variable names and similar. Unlike Windows, in Linux symbols are a plain text file typically called System.map found in /boot/.

³³ If stability is not an issue (ie you are about to ‘pull the plug’ anyway) an attempt at copying /dev/mem should be attempted. All tests, albeit not extensive, performed as research for this project showed no stability issues with using /dev/mem as input for dd.

The init task symbol can be found with:

```
$> cat /boot/System.map | grep init_task
c0479b2c r __ksymtab_init_task
c047ef28 r __kstrtab_init_task
c048cba0 D init_task
```

The line of interest it is the one with the “D”

Similar to how the offsets for parts of an EPROCESS structure were located for windows using the structure output from a windows debugger (Appendixes B-D), the Linux kernel source can be inspected to located similar part of the process structure in Linux.

The template for a task structure can be found in sched.h(about 1100 lines of code):

```
$>cat /usr/src/linux-KERNELVERSION/include/sched.h
```

A sample task_struct listing from a 2.6.10 kernel can be found at the end of this

Appendix. It is easy to see the similarity to the EPROCESS structure (related to the goals of this paper anyway).

Converting a Virtual Address to a Physical Address actually requires more steps than in Windows. Each task has a memory map struct, keeping track of virtual memory area structs. In Virtual Memory, there is a virtual file which has a pointer to a dentry which has a pointer to an inode which has a mapping into address space. This address space actually tracks page descriptors.

Research done during the creation of this text revealed that Mariusz Burdach actually has already created documentation similar to what has been proposed for Linux. Rather than replicate his work here, a citation is given to his work and it is up to the reader to study his work at their leisure. (Burdach, 2004).

Linux Task Structure (from sched.h kernel 2.6.10)

```

struct task_struct {
    volatile long state; /* -1 unrunnable, 0 runnable, >0 stopped */
    struct thread_info *thread_info;
    atomic_t usage;
    unsigned long flags; /* per process flags, defined below */
    unsigned long ptrace;

    int lock_depth; /* Lock depth */

    int prio, static_prio;
    struct list_head run_list;
    prio_array_t *array;

    unsigned long sleep_avg;
    long interactive_credit;
    unsigned long long timestamp, last_ran;
    int activated;

    unsigned long policy;
    cpumask_t cpus_allowed;
    unsigned int time_slice, first_time_slice;

#ifdef CONFIG_SCHEDSTATS
    struct sched_info sched_info;
#endif

    struct list_head tasks;
    /*
     * ptrace_list/ptrace_children forms the list of my children
     * that were stolen by a ptracer.
     */
    struct list_head ptrace_children;
    struct list_head ptrace_list;

    struct mm_struct *mm, *active_mm;

/* task state */
    struct linux_binfmt *binfmt;
    long exit_state;
    int exit_code, exit_signal;
    int pdeath_signal; /* The signal sent when the parent dies */
    /* ??? */
    unsigned long personality;
    unsigned did_exec:1;
    pid_t pid;
    pid_t tgid;
    /*
     * pointers to (original) parent process, youngest child, younger sibling,
     * older sibling, respectively. (p->father can be replaced with
     * p->parent->pid)
     */
    struct task_struct *real_parent; /* real parent process (when being debugged) */
    struct task_struct *parent; /* parent process */
    /*
     * children/sibling forms the list of my children plus the
     * tasks I'm ptracing.
     */
    struct list_head children; /* list of my children */
    struct list_head sibling; /* linkage in my parent's children list */
    struct task_struct *group_leader; /* threadgroup leader */

    /* PID/PID hash table linkage. */

```

```

struct pid pids[PIDTYPE_MAX];

wait_queue_head_t wait_chldexit;      /* for wait4() */
struct completion *vfork_done;        /* for vfork() */
int __user *set_child_tid;            /* CLONE_CHILD_SETTID */
int __user *clear_child_tid;          /* CLONE_CHILD_CLEARTID */

unsigned long rt_priority;
unsigned long it_real_value, it_prof_value, it_virt_value;
unsigned long it_real_incr, it_prof_incr, it_virt_incr;
struct timer_list real_timer;
unsigned long utime, stime;
unsigned long nvcsw, nivcsw; /* context switch counts */
struct timespec start_time;
/* mm fault and swap info: this can arguably be seen as either mm-specific or thread-
specific */
unsigned long min_flt, maj_flt;
/* process credentials */
uid_t uid, euid, suid, fsuid;
gid_t gid, egid, sgid, fsgid;
struct group_info *group_info;
kernel_cap_t cap_effective, cap_inheritable, cap_permitted;
unsigned keep_capabilities:1;
struct user_struct *user;
#ifdef CONFIG_KEYS
struct key *session_keyring; /* keyring inherited over fork */
struct key *process_keyring; /* keyring private to this process (CLONE_THREAD) */
struct key *thread_keyring; /* keyring private to this thread */
#endif
unsigned short used_math;
char comm[16];
/* file system info */
int link_count, total_link_count;
/* ipc stuff */
struct sysv_sem sysvsem;
/* CPU-specific state of this task */
struct thread_struct thread;
/* filesystem information */
struct fs_struct *fs;
/* open file information */
struct files_struct *files;
/* namespace */
struct namespace *namespace;
/* signal handlers */
struct signal_struct *signal;
struct sighand_struct *sighand;

sigset_t blocked, real_blocked;
struct sigpending pending;

unsigned long sas_ss_sp;
size_t sas_ss_size;
int (*notifier)(void *priv);
void *notifier_data;
sigset_t *notifier_mask;

void *security;
struct audit_context *audit_context;

/* Thread group tracking */
u32 parent_exec_id;
u32 self_exec_id;
/* Protection of (de-)allocation: mm, files, fs, tty, keyrings */
spinlock_t alloc_lock;
/* Protection of proc_dentry: nesting proc_lock, dcache_lock,
write_lock_irq(&tasklist_lock); */
spinlock_t proc_lock;

```



```

/* context-switch lock */
    spinlock_t switch_lock;

/* journalling filesystem info */
    void *journal_info;

/* VM state */
    struct reclaim_state *reclaim_state;

    struct dentry *proc_dentry;
    struct backing_dev_info *backing_dev_info;

    struct io_context *io_context;

    unsigned long ptrace_message;
    siginfo_t *last_siginfo; /* For ptrace use. */
/*
 * current io wait handle: wait queue entry to use for io waits
 * If this thread is processing aio, this points at the waitqueue
 * inside the currently handled kiocb. It may be NULL (i.e. default
 * to a stack based synchronous wait) if its doing sync IO.
 */
    wait_queue_t *io_wait;
#ifdef CONFIG_NUMA
    struct mempolicy *mempolicy;
    short il_next; /* could be shared with used_math */
#endif
};

```

Appendix J: GNU General Public License

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so

that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

- a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
- b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any

part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you

received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by

all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Cited Links:

.NET Framework FILETIME specification

<http://msdn2.microsoft.com/en-us/library/system.runtime.interopservices.comtypes.filetime.aspx>

/proc/kcore May Be Going Away. August, 2003.

http://www.kernel-traffic.org/kernel-traffic/kt20030908_229.html#5

AlphaServer Comparison Chart, December 1996

<http://h18002.www1.hp.com/alphaserver/archive/comp/dec96.html>

Forensic Analysis of a Linux System. Mariusz Burdach. April 4, 2004.

<http://www.securityfocus.com/infocus/1773>

Forensic. Merriam-Webster Online. <http://m-w.com/dictionary/forensics>

Johnson, Micheal. RedHat, Inc's Network Console and Crash Dump Facility. 2002.

<http://www.redhat.com/support/wpapers/redhat/netdump/>

KB 156280: How to use DumpChk to check a memory dump file

<http://support.microsoft.com/kb/156280/>

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

KB 254649: Overview of memory dump file options for Windows Server 2003, XP and 2000 <http://support.microsoft.com/kb/254649/>

KB 274598: Complete memory dumps are not available on computers that have 2 or more gigabytes of RAM

<http://support.microsoft.com/kb/274598/>

KB 307973: How to configure failure and recovery options in windows

<http://support.microsoft.com/kb/307973/>

KB 555223: RAM, Virtual Memory, Pagefile and all that stuff

<http://support.microsoft.com/default.aspx?scid=kb;en-us;555223>

Kurt Dillard. Rootkit Battle: Rootkit Revealer vs Hacker Defender. Aug 3, 2005.

http://searchwindowssecurity.techtarget.com/columnItem/0,294698,sid45_gci1112754,00.html

Michael Marxmeir, Database Performance Tuning. 2001.

<http://www.hp-eloquence.com/support/misc/dbtuning.html>

Rootkit Levels of Infection and Mitigation

http://searchopensource.techtarget.com/tip/1,289483,sid39_gcil149598,00.html

Six tips for efficient memory usage

<http://www.microsoft.com/whdc/driver/perform/mem-alloc.mspx>

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

Deleting May be easy, but your hard drive still tells all. Eric Taub. Apr 5, 2006. New York Times.

<http://www.nytimes.com/2006/04/05/technology/techspecial4/05forensic.html?ex=1152417600&en=089d847c6b92aa27&ei=5070>

Why you cant tread a FILETIME as an int64.

<blogs.msdn.com/oldnewthing/archive/2004/08/25/220195.aspx>

Windows Server 20003 Service Pack 1 Changes

<http://technet2.microsoft.com/WindowsServer/f/?en/Library/c8f4d2ac-29b8-4546-8db5-5fa22f0083791033.mspx>

Windows Server 2003 Service Pack 1 Changes: \Device\PhysicalMemory Object

<http://technet2.microsoft.com/WindowsServer/en/Library/e0f862a3-cf16-4a48-bea5-f2004d12ce351033.mspx?mfr=true>

Cited Works

Andrew Tanenbaum. *Operating Systems: Design and Implementation* Second Edition. 1997.

Bill Nelson, Amelia Phillips, Frank Enfinger, Chris Steuart. *Guide to Computer Forensics and Investigations*. Thomson Course Technology. 2004.

Brian Carrier, Joe Grand. A Hardware-Based Memory Acquisition Procedure for Digital Investigations. *Digital Investigation Journal*. February 2004.

Bruce Middleton. *Cyber Crime Investigator's Field Guide*, Second Edition. Auerbach. April 2005.

Caloyannides, Michael A. *Computer Forensics and Privacy*. Artech House, Inc. 2001.

Corbató, F. J., and V. A. Vyssotsky, Introduction and overview of the Multics system, *AFIPS Conf Proc* 27, 185-196, 1965.

D. Brezinski, T. Killalea. RFC 3227: Guidelines for Evidence Collection and Archiving. February 2002.

Dan Farmer, Wietse Venema. *Forensic Discovery*. Addison Wesley Professional 2005.

Debra Little, John Shinder, Ed Tittel. *Scene of the Cybercrime: Computer Forensics Handbook*. Syngress. September 2002 .

Digital Forensics Research Workshop. "A Road Map for Digital Forensics Research" 2001. www.dfrws.org

George Mohay, Alison Anderson, Byron Collie, Olivier de Vel, and Rodney D. McKemmish. *Computer and Intrusion Forensics*. Artech House. 2003.

Greg Hoglund, James Butler. *Rootkits*. Addison Wesley. 2006.

Grugq. Remote Execution of Binary without creating a file on disk. Phrack #62.

Information and Communications Security: 6th International Conference, ICICS 2004, Malaga, Spain, October 27-29, 2004. Proceedings.

Jim Chow, Ben Pfaff, Tal Garfinkel, Kevin Christopher, Mendel Rosenblum. *Understanding Data Lifetime via Whole System Simulation*. 2004.

Jim Chow, Ben Pfaff, Tal Garfinkel, Mendel Rosenblum. Shredding Your Garbage: Reducing Data Lifetime Through Secure Deallocation. 14th Usenix Security Symposium. 2005.

John L. Hennessy, David A Patterson. Computer Architecture A Quantitative Approach. Third Edition. Morgan Kaufmann, 2003.

John R. Vacca. Computer Forensics: Computer Crime Scene Investigation. Charles River Media. 2002.

Joseph Grand. Memory Imaging and Forensic Analysis of Palm OS Devices. At Stake. 2002.

Lynn, Michael. Blackhat Breifings (unedited): USA 2005. Las Vegas, NV. July 27-28, 2005. Proceedings.

Mandia, Kevin, Prorise, Chris. Incident Response & Computer Forensics. McGraw Hill. 2003.

Marc Rodgers. West Coast Security Forum 2004. , Purdue. 2004.

Mark E Russinovich, David A. Solomon. Microsoft Windows Internals. Fourth Edition. Microsoft Press. 2005.

Michael A. Caloyannides. Privacy Protection and Computer Forensics, Second Edition. Artech House. 2004.

Microsoft Corp. Debugging Tools for Windows help file. Microsoft Corp. January 6, 2006.

Mike Schroeder and Jerry Saltzer "A hardware architecture for implementing protection rings" at the *Third ACM Symposium on Operating System Principles* in Palo Alto, CA, in October 1971.

Pluf and Ripe. Advanced Anti Forensics – SELF. Phrack #63.

Reith, Carr, Gunsch. An Examination of Digital Forensic Models. International Journal of Digital Evidence. Vol 1, Issue 3. Fall 2002.

Richard Nolan, Colin O’Sullivan, Jake Branson, Cal Waits. Carnegie Mellon University. March 2005.

Simon Baker, Patrick Green, Thomas Meyer, Garaidh Cochrane. Checking Microsoft Windows for Signs of Compromise. Ver 1.3.4. Oct 28, 2005.

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

Wayne Jansen, Rick Ayers. Guidelines on PDA Forensics. Recommendations of the National Institute of Standards and Technology. US Department of Commerce Special Publication 800-72. November 2004.

Willis Ware. Security Controls for Computer Systems. Department of Defense. February 11, 1970.