Australian Government
Department of Defence
Defence Science and
Technology Organisation

Memory Forensics: Review of Acquisition and Analysis Techniques

Grant Osborne

Cyber and Electronic Warfare Division

Defence Science and Technology Organisation

DSTO–GD–0770

ABSTRACT

This document presents an overview of the most common memory forensics techniques used in the acquisition and analysis of a system's volatile memory. Memory forensics rose from obscurity in 2005 in response to a challenge issued by the Digital Forensics Research Workshop (DFRWS). Since then, investigators and researchers alike have begun to recognise the important role that memory forensics can play in a robust investigation. Volatile memory, or Random Access Memory (RAM), contains a wealth of information regarding the current state of a device. Memory forensics techniques examine RAM to extract information such as passwords, encryption keys, network activity, open files and the set of processes and threads currently running within an operating system. This information can help investigators reconstruct the events surrounding criminal use of technology or computer security incidents.

Published by

DSTO Defence Science and Technology Organisation
PO Box 1500
Edinburgh, South Australia 5111, Australia

Telephone: 1300 DEFENCE
Facsimile: (08) 7389 6567

AR No. 015-765
November, 2013

APPROVED FOR PUBLIC RELEASE

Memory Forensics: Review of Acquisition and Analysis Techniques

Executive Summary

In this document the state of the art in memory forensics is examined. Memory forensics is a domain of digital forensics focused on the investigation of information stored in a system's volatile memory (or RAM). RAM contains a wealth of information such as logged in users, process objects, threads, network connections and open files. In both criminal cases and security incident responses, memory forensics can provide useful insights into the state of a system under investigation.

First the mechanisms by which Windows operating systems organise memory are discussed. Key areas discussed include the methods involved in memory management, virtual address translation and the Windows data structures that are used to manage processes and threads. This information is useful in understanding the analysis techniques used by memory forensics tools.

Common methods for acquiring exact copies of RAM were examined next. It was found that hardware-based approaches were reliable, but had low availability. Conversely, software-based acquisition approaches were found to be more commonly available, but produced images of memory with less reliability.

Methods to analyse RAM images are discussed. A literature review found that the most commonly extracted items of information include lists of currently executing processes and threads, cryptographic keys, registry information, established network connectivity, command history and open files. Also discussed was the implementation of these techniques in several tools such as the Volatility Framework.

This review identified that there is no single best technique for the acquisition of memory samples from a target system. An investigator needs to make a judgement on the best approach to take in the current situation. Overall the literature shows that that memory forensics techniques are useful for extracting the current state of a system. Such system state information could be useful for Defence in a security incident response.

THIS PAGE IS INTENTIONALLY BLANK

Author

Dr Grant Osborne
CEWD

Grant was awarded a Bachelor of Advanced Computer and Information Science (Honours) from University of South Australia in 2006. Upon graduation he commenced work as a software engineer for a major Australian defence contractor. In 2008 he commenced a PhD at the University of South Australia, which he completed in 2011. His experience includes over 5 years of software engineering developing both desktop and web based applications. Grant has worked as a researcher in the Cyber Assurance and Operations (CAO) branch at DSTO since 2012. His research interests include visual analytics, information visualisation and computer security.

THIS PAGE IS INTENTIONALLY BLANK

Contents

Figures

Tables

THIS PAGE IS INTENTIONALLY BLANK

Acronyms

AES Advanced Encryption Standard.

API Application Programming Interface.

AWE Address Windowing Extensions.

BIOS Basic Input/Output System.

CCV Card Code Verification.

CPU Central Processing Unit.

CSRSS Client/Server Runtime Subsystem.

DES Data Encryption Standard.

DFRWS Digital Forensic Research Workshop.

DKOM Direct Kernel Object Manipulation.

DLL Dynamic-Link Library.

DMA Direct Memory Access.

FBI Federal Bureau of Investigation.

FDE Full Disk Encryption.

GDI Graphics Device Interface.

GUID Globally Unique Identifier.

IP Internet Protocol.

MMU Memory Management Unit.

NAS Network-attached Storage.

PAE Physical Address Extensions.

PCB Process Control Block.

PCI Peripheral Component Interconnect.

PDB Program Database.

PDE Page Directory Entry.

PEB Process Environment Block.

PGP Pretty Good Privacy.

PTE Page Table Entry.

PTI Page Table Index.

RAM Random Access Memory.

RPC Remote Procedure Call.

SAN Storage Area Network.

SATA Serial Advanced Technology Attachment.

SSD Solid State Drive.

SSL Secure Sockets Layer.

TEB Thread Environment Block.

UMA Upper Memory Area.

URL Uniform Resource Locator.

USB Universal Serial Bus.

VAD Virtual Address Descriptor.

VM Virtual Machine.

Glossary

CR3 A control register for an x86 processor. The CR3 register enables the processor to translate virtual addresses into physical addresses by locating the page directory and page tables for the current task.

EPROCESS An Executive Process, or EPROCESS data structure, is a Windows API data structure that represents a process.

ETHREAD An Executive Thread, or ETHREAD data structure, is a Windows API data structure that represents a task.

KTHREAD A Kernel Thread data structure is used by the Windows Kernel to schedule and synchronise running threads.

THIS PAGE IS INTENTIONALLY BLANK

1 Introduction

Digital forensics is a branch of forensic science primarily concerned with the acquisition and analysis of material found on computer devices used for inappropriate or illegal purposes. These purposes include hacking, cyber stalking, identity theft, fraud and the production and procurement of child exploitation material. In addition to criminal investigations, digital forensic investigation techniques have been adopted for the purpose of security incident response and management of business-critical systems. In short, the goal of digital forensics is to capture and analyse binary data stored or transmitted by an electronic device in order to help support or refute a theory on how the misuse of equipment or criminal activity occurred [1]. This captured binary data is more commonly known as digital evidence. Common types of digital evidence investigated include images, text, video and audio files [1].

To date, digital forensic investigations have focused on the acquisition and analysis of persistent data. Persistent data is information contained on hard disk drives, storage media such as Universal Serial Bus (USB) flash drives or optical disk storage formats such as DVDs. Investigation procedures involve powering off the target machine, creating an identical image of the attached storage media and performing a postmortem analysis of the collected information [2].

Similar "pull the plug" procedures have been advocated throughout the digital forensics domain, in particular by U.S departments including the National Institute of Justice [3] and the National Institute of Standards and Technology [4]. Although these processes are valuable for first responders, they commonly neglect the acquisition of Random Access Memory (RAM) [5], even though the literature suggests that data be collected based on the order of volatility (i.e. collect the most volatile source of data first) [1, 6].

Persistent data investigations face difficult challenges in handling developments in hard drive capacity, Full Disk Encryption (FDE) and the rising prevalence of in-memory malicious software [7]. The increase in applications designed to leave little to no persistent trace of their activities (such as Mozilla Firefox with private browsing enabled) also presents a major challenge for current investigative techniques that rely on traces of user actions being left on the persistent media. Current tools and techniques do not scale effectively with increasing hard drive capacities and are completely ineffectual when analysing storage with encryption or software that leaves little to no permanent trace.

The ever-growing storage capacity of hard drives dramatically increases the time taken for both acquisition and analysis of the information they contain [8]. Imaging a hard drive of only 250 gigabytes, which by today's standards is relatively small, may take upwards of 100 minutes [9]. Recent disk and transfer speed improvements have been made through technologies such as Solid State Drives (SSDs), Serial Advanced Technology Attachment (SATA) Express, Thunderbolt and USB 3.1. However, adoption of these technologies is not yet prevalent in a majority of machines under investigation. Furthermore, the sheer amount of data in high capacity Network-attached Storage (NAS) or Storage Area Networks (SANs) means that even with speed improvements it is still highly unlikely that all of the data could be analysed in a reasonable time frame.

FDE is another major challenge for techniques focused on persistent data oriented analysis [7]. Such technologies may be used to prevent an investigator access to all of the digital evidence stored on a disk [10]. Many freely available and simple to use software products such as TrueCrypt[1]

[1] htpp://en.wikipedia.org/wiki/TrueCrypt

or Pretty Good Privacy (PGP)[2] and the inclusion of encryption technologies as security features in popular operating systems (such as Windows 7's BitLocker[3]) are causing the prevalence of FDE technology to increase. Defendants are often not compelled to give up encryption keys to investigators. Furthermore, a suspect can feign cooperation by providing only one of many keys to multiple containers or by providing an incorrect key to investigators. In the first case, the suspect may provide a key to a container with no incriminating evidence. In the latter case, the defendant could declare that the container had somehow been damaged as part of the investigation. Casey et al. [10] cite several cases that have been foiled by FDE including the case of the Brazilian banker, Daniel Dantas. Dantas used TrueCrypt to prevent access to hard drives captured from his apartment by Brazilian Police. To date, neither dictionary-based attacks from the Brazilian National Institute of Criminology nor attempts by the Federal Bureau of Investigation (FBI) have successfully accessed the data.

Memory resident malicious software (malware) presents a challenge to "live" investigation digital forensic analysis techniques [7]. Malware may compromise the reliability of an investigation by altering the perceived system state (such as by providing a fake list of running processes to the investigator). Furthermore, malware may even be used to execute anti-forensics techniques that modify the data captured by digital forensics tools and techniques [11]. Malware authors often employ compression, armouring and obfuscation techniques that make static analysis and reverse engineering of their code incredibly difficult [12]. In the case of commonly known malware such as Code Red, Witty and SQL Slammer, evidence of execution exists only within volatile memory [13–15]. As such, current digital forensics approaches may fail to find evidence of this malicious software by performing postmortem analysis after the machine has been powered off [16].

There has been a research focus shift toward the recovery of transient system-state information stored in volatile memory, most commonly referred to as memory forensics [17, 18]. Memory forensics typically seeks information that only exists in RAM to provide insight into the state of a system, to corroborate any existing evidence and to provide helpful synergies to persistent data investigations. The information it extracts commonly includes encryption keys, passwords, active network connections, open files and currently executing processes (including hidden processes and malware). An example of how memory forensics can help persistent data analysis could be through the identification of encryption keys in memory. These keys could be used to enable investigators to perform persistent data investigations on encrypted containers or devices with FDE.

Memory forensics as a research domain has been gaining momentum since the 2005 Digital Forensic Research Workshop (DFRWS)'s "Memory Forensics Challenge". This challenge caused a significant research effort focused on the analysis of the contents of memory dumps, with tools such as `memparser` by Betz [19] and `kntlist` by Garner [20] emerging in direct response. Unfortunately these initial tools had major drawbacks. One drawback was the use of hard coded memory structures to extract target memory artifacts. Another drawback was the reliance on lists maintained by the Windows kernel in order to retrieve processes, threads and other items that exist in the memory sample.

Building on these first steps, techniques such as those from Schuster [21] showed that it was possible to use knowledge of the in-memory structure of processes and threads to create search patterns that enable entire memory dumps to be analysed for traces of these objects. These techniques helped to address issues associated with enumerating lists kept by the Windows kernel

[2] htpp://en.wikipedia.org/wiki/Pretty_Good_Privacy
[3] http://en.wikipedia.org/wiki/BitLocker_Drive_Encryption

such as the failure to detect objects that have been hidden by manipulating the kernel process lists. However, these search patterns are also limited as they require prior in-depth knowledge of the characteristics of the operating system being analysed [22]. As such, the current research direction within the memory forensics domain is focused on the analysis of memory dumps without relying on hard coded structures [17, 22].

The domain of memory forensics can be considered a sub-domain of live forensics. Live forensics, as the name implies, attempts to capture as much information as possible from a system that is currently running [13]. Live forensics tools commonly rely on the underlying system in order to acquire information such as by copying files, or by examining the task manager to view the list of running processes. More recently, first responders who perform live forensics have begun to include memory forensics in their suite of triage tools [8, 11]. This is because memory forensics tools have been found to be less destructive than executing new processes on the system in order to capture data [13].

This literature review provides an overview of the most common techniques used to acquire and then analyse a computer's RAM. The primary focus of this review is on techniques that are used to analyse the Windows family of operating systems. This scope is based on the assumption that investigators are most likely to encounter Windows-based machines in practice, owing to the high popularity and market positioning of the operating system. This literature review is structured as follows: Section 2 presents a brief technical overview of the main data structures used to manage the execution of processes and threads within a system's RAM; Section 3 provides an analysis of the most common methods of memory acquisition; Section 4 discusses the most common techniques used for the analysis of memory captures; and Section 5 provides a summary of the literature presented.

2 Technical Background

This section provides an overview of the processes involved in memory management and the in-memory structures pertinent to the acquisition and analysis of physical memory. This section is largely based on content from Russinovich, Solomon and Ionescu's "Windows Internals" book [23]. A deeper understanding of Windows memory management, processes and threads can be acquired by reading the works of these authors. Section 2.1 provides an overview of memory management. It describes the mechanisms involved in virtual memory, address translation and memory paging. Section 2.2 describes the Executive Process EPROCESS and Executive Thread ETHREAD structures, commonly targeted by memory analysis to understand the current state of a system from a memory capture.

2.1 Memory Management Overview

Processes running on modern multitasking operating systems operate on an abstraction of RAM, called virtual memory [7]. In these systems, processes are given the impression that they are operating with a large contiguous section of memory. The reality however, is that a process' memory may be spread across non-contiguous pages of physical memory, or may have even been paged out to a backup storage device (typically the hard drive). This abstraction requires specific hardware support via a Memory Management Unit (MMU) and offers inherent advantages over

directly addressing RAM. These advantages include providing each process with its own protected view of system memory, as well as monitoring and restricting read/write activities through the use of privilege rules [24]. The layouts of virtual and physical memory can differ as illustrated in Figure 1 (based on [23] p. 15). Notice that the blocks of virtual memory do not map to contiguous physical addresses.

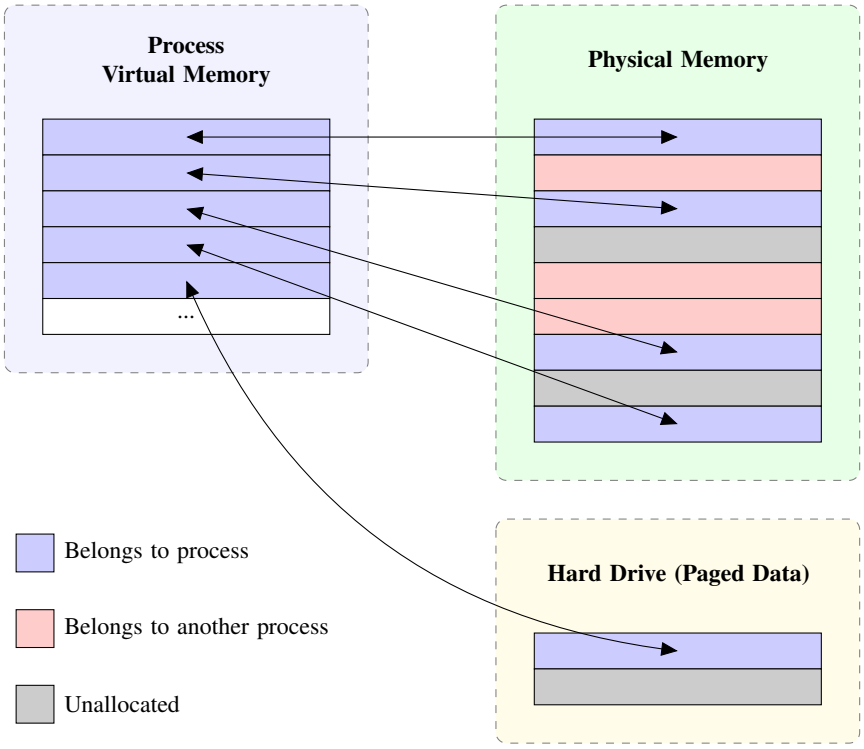

Figure 1: *Mapping virtual memory pages to physical memory pages*

Each process has its own private virtual address space that is non-accessible to other processes, unless a portion of memory is specifically shared [23]. This stops potential collisions in memory and also helps to prevent privilege violations between different executables. 32-bit x86 user-mode processes have access to 2GB of virtual memory by default [23], with the other half of the total (4GB) address space reserved for system usage (assuming advanced features such as Physical Address Extensions (PAE), Address Windowing Extensions (AWE) or large address spaces are turned off). Kernel memory is shared between, and available to all, system applications and components. As such these applications are designed with particular care to ensure they do not cause system instability [23]. Furthermore, this system memory space contains critical regions, such as the process page tables required for virtual-to-physical address translation, as well as system pool data that contains volatile system state information [23].

As discussed, each process operates within its own virtual memory address space. In order to manipulate physical data the MMU must continuously translate, or map, a process' virtual addresses into physical addresses. At the hardware level, memory storage is organised into pages. Each page is commonly 4 kB on x86 systems [23, 24]. The operating system maintains a page directory that stores Page Directory Entry (PDE) pointers. PDE pointers are 4 bytes each and the entry may contain up to 1024 unique links to page tables. Page tables, in turn, store up to 1024

pointers to Page Table Entry (PTE) data structures that correspond to a page in memory. As such, the translation from virtual to physical memory addresses requires two main steps. The MMU first recovers the base address of the page directory, which is stored in the CR3 register of the system's processor. The first 10 bits of a virtual address can then be used as an index into the page directory to retrieve the desired PDE. The PDE in conjunction with the Page Table Index (PTI), which is the next 10 bits of the virtual address, are used to identify the appropriate page table and PTE. Finally, the appropriate page in RAM is determined by following the link in the PTE.

The virtual address translation mechanism described previously assumed that the data was immediately available in physical memory. It is often the case however, that the total amount of virtual memory consumed by the currently executing processes on a system is larger than the size of the physical memory [7]. Operating systems cope with this scenario by temporarily swapping memory contents out to the hard drive (known as paging) in order to free up the required space in physical memory. When a thread (the working unit of a process) attempts to operate on a swapped-out page, the MMU generates a page fault interrupt to request that the operating system transfer the requested page back into physical memory. The current location of a virtual address (hard disk or memory), is indicated within the PDE and PTE data structures of a process by setting a valid flag to 1 or 0. There are other flags within these structures that specify whether the memory page is currently in transition or is a non-swappable address. By default the page file is stored in the operating system installation root, in a file called `pagefile.sys` [23].

This section has provided an overview of the elements of memory management in Windows family operating systems. In the next section the main data structures targeted by memory forensics techniques are examined.

2.2 Processes and Threads

The Windows family of operating system records all of the metadata necessary to manage the processes currently being executed in physical memory [21]. Even if a process exits its metadata may be retained in memory for weeks [25]. The information stored in the metadata provides a snapshot of the processes and threads that are either currently or have recently executed on a system. As such an understanding the common data structures utilised by Windows operating systems to manage the execution of processes is an important part of memory analysis. Figure 2 (based on [23] p. 336) illustrates the process and thread data structures and their relationships with each other in a simplified manner.

A few key aspects of Figure 2 are discussed here. Processes are represented in memory by an EPROCESS data structure [23]. The EPROCESS block contains many process attributes, as well as pointers to a number of related data structures. For example, a process has one or more threads represented by an ETHREAD block. The EPROCESS block and the data structures it contains reside within the system address space. Another key structure, the Process Environment Block (PEB), exists in process address space. The PEB contains a process's global contextual information such as start up parameters or synchronisation objects. Each ETHREAD block has a pointer to a KTHREAD block. The KTHREAD block has a pointer the Thread Environment Block (TEB) which maintains user-mode contextual information about threads. The key components of the EPROCESS, ETHREAD and KTHREAD data structures will now be examined.

EPROCESS. The EPROCESS block is an opaque Windows Application Programming Interface (API) data structure whose primary purpose is to represent process objects for internal

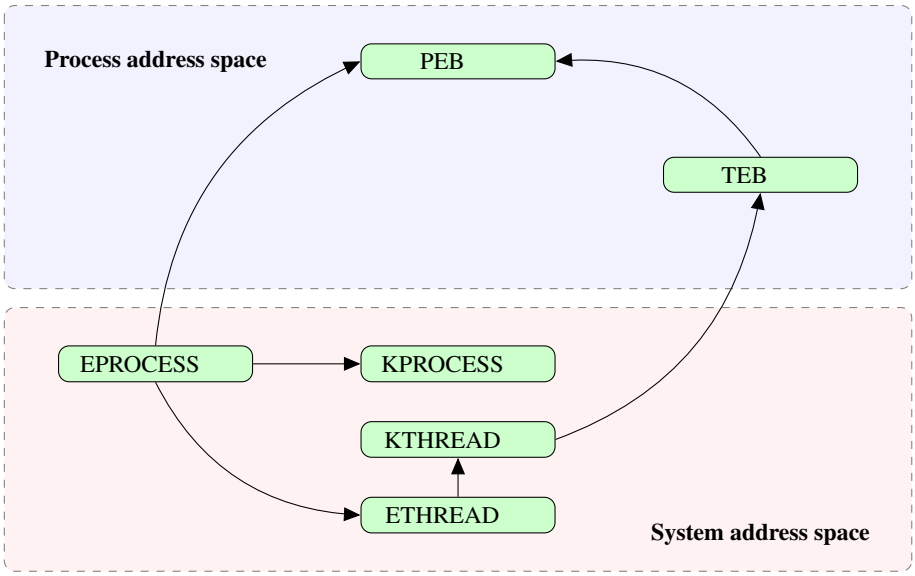

Figure 2: *Data structures associated with processes and threads*

Windows operating system functions [26]. Table 1 (based on [23]) examines the components of the data structure relevant to memory forensics. Further details of the EPROCESS structure, including memory offsets and other technical details can be found in "Windows Internals" [23].

Table 1: *`EPROCESS` block key components*

Element	Purpose
Kernel Process Block	Used to record the amount of time a process spends on the Central Processing Unit (CPU) or within kernel/user-mode, the process' affinity, scheduling priority, process lock status and many other items.
Process Identification	Unique identifier for the process and its parent.
Quota	Stores the limits on processor usage. The structure also contains flags for which type of memory to be stored in (non paged or paged).
Virtual Address Descriptor	Describe the status of the virtual portions of the address space that exist in the process.
Virtual Memory Information	Current and peak virtual size of the process, page file usage and the page table entry for the process page directory.
Handle Table	Contains the address to the per-process handle table, which manages handles to the objects the process is using.
Process Environment Block	Contains information about the image that is running (such as its base address and version number), process heap information and thread-local storage utilisation.
Windows Subsystem Process	A structure that contains details needed by the kernel-mode component of the Windows subsystem for a given process.

ETHREAD. In Windows operating systems threads are represented by an ETHREAD block [23, 26]. Similar to the EPROCESS block, the ETHREAD block and the structures it points to exist in system address space. As per the EPROCESS block, the Client/Server Runtime Subsystem (CSRSS) also maintains a structure for each thread created in a Windows subsystem application. Finally, for threads that have called a User or Graphics Device Interface (GDI) function the kernel-mode Windows subsystem (`Win32k.sys`) maintains a per-thread structure that the ETHREAD

points to. Table 2 (Based on [23]) highlights the pertinent components of the ETHREAD data structure for memory forensics.

Table 2: *`ETHREAD` block key components*

Element	Purpose
KTHREAD	Contains the information that the Windows kernel needs to perform thread scheduling and synchronisation on behalf of running threads.
Thread Time	The creation and exit time of the thread.
Process Identification	Contains the process ID and a pointer to the EPROCESS block that the thread belongs to.
Input/Output Information	A list of pending input and output requests.
Start Address	The address of the thread start routine.

KTHREAD. The KTHREAD contains scheduling and synchronisation information that is used by Windows to maintain a process' threads. The key elements of the KTHREAD structure are outlined in Table 3 (Based on [23]). One particularly useful component of the KTHREAD is its pointer to the TEB. The TEB is a process space data structure that stores contextual information for the thread, including information about the executable image loader and any Dynamic-Link Library (DLL) handles the thread has [23].

Table 3: *`KTHREAD` block key components*

Element	Purpose
Execution Time	This records the total user and kernel-mode CPU time for the thread.
Cycle Time	This records the total amount of time spent on the CPU.
Kernel Stack Pointer	The thread stores a pointer to the upper and base address of the kernel stack.
Scheduling Information	Stores the base and current priority of the thread, its affinity mask as well as the thread's ideal processor, deferred processor and the next processor it will execute on.
TEB	Contains a Thread ID, a pointer to the PEB and also information about whether the thread is using Windows Sockets, OpenGL, Remote Procedure Call (RPC), GDI and other user-mode code.

2.3 Summary

This section has provided an overview of the key memory management mechanisms and data structures that are of interest in memory forensics investigations. Firstly, the concept of virtual memory was introduced. This examined the way modern operating systems abstract RAM by using virtual memory mapping to provide processes with a seemingly contiguous allocation of memory, even though the physical mapping of these memory addresses may be scattered throughout RAM or paged into the system's page file. The next section presented several data structures that can be extracted from a memory image to provide a representation of the current state of a machine. These structures are the EPROCESS, ETHREAD and KTHREAD blocks. Each contains a wealth of information about the processes that are executing within a Windows operating system environment and can be used to locate open files and the DLLs that a process may be using. It is by exploiting these data structures (in conjunction with standard searches for strings and file carving techniques)

that analysts are able to reconstruct the state of a machine when examining a memory image. The following section examines the methods commonly used to acquire memory images.

3 Memory Acquisition

Techniques used to extract volatile memory images from target systems are defined as either hardware-based or software-based solutions. Software-based solutions rely on the operating system in order to perform memory capture. Hardware-based solutions in contrast, directly access the volatile memory of the target system. To date, hardware-based solutions for memory acquisition have been considered the most reliable as it is difficult to obtain a complete and accurate memory snapshot from software [7].

In order to measure the efficacy of acquisition techniques, Schatz [27] suggests several criteria. These are:

- the fidelity and reliability of the generated image
- the availability of the mechanism (software or hardware) used to capture the image.

The *fidelity* and *reliability* criteria ensures that the image obtained is a "precise copy of the host's memory" [27]. In particular these criteria dictate that the resultant memory image should be trustworthy (i.e. the capture process is not interfered with by malware or other actors). Schatz [27] suggests that if the result is not guaranteed to be trustworthy there should be no memory capture at all. This is because if the memory image contains misinformation as a result of tampering, the information gained from its analysis is likely to add confusion to an investigation.

The *availability* criteria stipulates that the technique must work on arbitrary computers and/or devices — essentially meaning that the method be operating system agnostic and not require specialised techniques.

Vömel and Freiling [7] suggest slightly different criteria used to measure efficacy for an acquisition technique: atomicity and availability. The *availability* factor extends Schatz's [27] definition, stating that a technique that is characterised by a high availability does not make any assumptions about particular pre-incident preparatory measures and can be applied without knowledge of the execution environment and without requiring that any pre-configurations exist prior to its execution. The *atomicity* of a technique reflects the demand to produce an accurate and complete image of a host's volatile storage, which encompasses the fidelity and reliability requirements put forth by Schatz [27].

The remainder of this section will analyse a variety of memory acquisition techniques, using the factors of atomicity and availability as a basis for discussion. For reference, an ideal acquisition method would be characterised as both highly atomic and highly available [7]. Section 3.1 discusses hardware-based acquisition methods; Section 3.2 discusses software-based acquisition techniques; and Section 3.3 examines cold booting techniques.

3.1 Hardware-Based Techniques

This section describes hardware-based techniques for memory sample acquisition. It is structured as follows: Section 3.1.1 describes approaches that require the installation of a dedicated devices; and Section 3.1.2 discusses approaches that acquire memory samples using hardware bus connections.

3.1.1 Dedicated Hardware

Dedicated hardware techniques are those that involve the installation of a physical device in order to assist investigators in acquiring a memory image from a target system. Carrier and Grand's [16] Tribble is an example of one such device. It is a Peripheral Component Interconnect (PCI) card that enables the capture of physical memory using Direct Memory Access (DMA) at the push of a button [16]. A major advantage of a device such as Tribble is that it is able to obtain a precise copy of physical memory without any interaction with the operating system running on the target machine. As such, it will bypass any subverted processes or memory structures running on the host machine [16]. The Tribble device itself must be installed prior to an investigation. When it is activated, the host machine is suspended to prevent any malicious code being executed during memory imaging. Once the memory dump is completed control is returned to the host operating system.

Until recently hardware devices such as Tribble were characterised as producing accurate and concise (highly atomic) memory images. This was based upon the assumption that as dedicated hardware does not rely on the operating system on the target machine it is able to produce a true picture of a system's memory. This includes any malware or rootkits that may be resident only in the system's volatile memory. However, Vömel and Freiling [7] discuss recent experiments that show that the northbridge chipset of motherboards is able to be reprogrammed to provide a subverted view of the system's memory, allowing regions to be swapped in and out with both software and hardware processes alike oblivious to the substitution. Although it is currently unlikely, investigators will need to consider that the hardware itself has been compromised when taking a memory snapshot [28]. This means that while hardware-based acquisition techniques generally produce highly atomic memory images, it is not necessarily always the case.

Dedicated hardware solutions for memory acquisition generally have low availability. For instance Tribble requires that the PCI card be installed in the system prior to use [16]. Such devices are not designed to be part of a first responder's triage toolkit, but assume that they will most likely be installed on critical infrastructure where high-stakes intrusions may occur. Dedicated hardware devices may also be used in a honeypot to facilitate learning about the tools and tactics utilised by attackers [7, 16].

3.1.2 Hardware Bus

A hardware bus facilitates data transfer between components (such as PCI) or between devices (such as USB or FireWire) within computer architectures. Memory acquisition techniques have been developed to exploit the use of the FireWire hardware bus to access the volatile memory of a system [29]. These approaches initially targeted Mac OS X and Linux-based systems, although they have also been shown to work on Windows operating systems [30]. FireWire-based approaches have shown to be quite popular primarily because the bus provides DMA by design. As such,

several proof-of-concept applications able to extract raw physical memory from a system through the FireWire bus have been developed [7].

Compared to dedicated hardware, hardware bus acquisition techniques are much more highly available. This is because hardware bus ports such as FireWire are quite common across both portable and desktop computers. However, Vidstrom [31] illustrates that when FireWire methods of acquisition access the Upper Memory Area (UMA) region of memory they can cause random system crashes. This decreases the reliability and atomicity of the results of this hardware bus approach. Other researchers have also found that the memory images captured through FireWire are often corrupt (i.e. missing data) [7]. As such, hardware bus approaches can have low atomicity.

3.2 Software-Based Techniques

This section describes software-based techniques for memory sample acquisition. It is structured as follows: Section 3.2.1 describes the extraction of memory using virtualised environments; Section 3.2.2 discusses the use of crash dumps to acquire memory samples; Section 3.2.3 explores low privilege user-mode applications; Section 3.2.4 discusses high privilege kernel-mode applications; and finally, Section 3.2.5 discusses the use of operating system injection as a technique for acquiring memory samples.

3.2.1 Virtualisation

Virtualisation provides isolated and reliable emulated system environments (Virtual Machines (VMs)) that execute within a host computer [32]. VMs are monitored to ensure proper management, sharing and restriction to the available hardware resources. Each VM is equipped with a virtual processor, memory, graphics adapters, network and IO interfaces and may run in parallel with many other VMs.

An important characteristic of VMs, is that they are capable of having their execution paused or suspended. The state of the machine is temporarily frozen and its virtual memory is saved to a hard disk on the underlying host. In the case of VMWare-based VMs, all of this volatile memory data is stored to a `.vmem` or `.vmss` file located in the working directory of the guest machine[4]. As such, the entire memory contents of such VMs can be acquired by simply suspending then copying this generated snapshot of main memory.

Within an environment that makes use of virtualisation, memory acquisition is both highly atomic and readily available [7]. This makes virtualisation-based approaches a highly useful testing ground for both memory analysis and memory acquisition techniques. Any techniques developed to acquire memory can do a bit-by-bit comparison of the data they captured with the `.vmem` file of the VM. This can help to verify and validate the technique that has been developed.

3.2.2 Crash Dumps

Windows operating systems can be configured to write memory dumps to a file when the system unexpectedly stops working [23]. In the case of a critical system failure the system state is frozen

[4] http://www.vmware.com/support/ws55/doc/ws_learning_files_in_a_vm.html

and the main memory as well as relevant CPU information are saved in the system root directory for later examination [7]. These dump files can then be examined with Microsoft's Debugging Tools for Windows or manually analysed [21]. For a memory forensics investigation, a responder may force the generation of a software crash dump by interrupting key system services using a third-party application, or by editing the registry to enable `Right-Ctrl+ScrollLock+ScrollLock` crash dumps [23]. This technique is only suitable to limited situations as the Windows registry must be modified in order to enable the manual crash dump technique. Furthermore, crash dumps can override parts of the system page file, which can decrease the total amount of evidence available [23].

3.2.3 User-Mode Applications

It is an extremely challenging task to atomically read physical memory using a user-mode application. Early attempts were able to access the `\\.\Device\PhysicalMemory` address to gain DMA on a Windows system. However, due to security reasons user-mode access to this object was restricted in Windows Server 2003 and later [7]. As such this technique is no long viable for memory acquisition and researchers have had to develop new techniques for acquisition, commonly involving the execution of another process on the target system.

PMDump [33] was developed to help investigators analyse the memory space of a target system. This tool only extracts the address space of a single process from volatile memory. This has the advantage of completing execution much more quickly and only capturing the memory space of a process of interest [33]. However, tools such as this must also run as a process within the operating system meaning that they modify the volatile memory of the system from which they are capturing process memory space. Furthermore, PMDump requires the process ID of the process the user wishes extract data from. This is not useful in the situation where the process has hidden itself from the operating system, such as in the case of rootkits.

A key advantage of user-mode applications is that they are characterised by a high level of availability. Such tools can be executed from an external USB flash drive to minimise system impact and be designed such that they will run on most Windows-based operating systems. The primary disadvantages of pure software based approaches is that by the very nature of being a software program, it must be loaded into volatile memory to execute [23]. As such, user-mode applications are not atomic when they acquire memory from a target system. Furthermore, as these applications rely on functions provided by the operating system they are vulnerable to subversion by malicious software. For instance, a rootkit may deny direct access to physical memory and return a modified representation during image generation [7]. As such, the overall atomicity of such techniques is questionable especially in the case of acquiring memory from a target system that may be running malware.

3.2.4 Kernel-Mode Applications

In order to mitigate the shortcomings of user-mode applications, software vendors and researchers are increasingly focusing on developing kernel-mode applications and drivers that can be used to create forensic copies of volatile memory [7]. These techniques are utilised in some freely available

tools, such as Memory DD[5], Windows Memory Toolkit[6] (Community Edition) and Memoryze[7]. Commercial software vendors have also developed some alternatives. These include Guidance Software's WinEN (Part of EnCase 6.11 and higher [8]), GMG Systems' KnTDD[9] and Fast Dump Pro from HB Gary [10].

Kernel-mode applications still suffer from the inherent weaknesses that affect user-mode applications [7]. Even if an application is a kernel-mode driver, it still modifies memory as Windows will create new process and thread structures when it is executed. Such techniques are also susceptible to compromised operating system functionality as per user-mode applications. Furthermore, kernel-based approaches suffer from availability issues as they require the investigator to have administrator privileges to install a driver-based approach onto the system itself prior to utilising it, or to execute a process that requires elevated user privileges. It could be argued that in practice this might not be an issue, due to the common practice of Windows users having administrative privileges.

3.2.5 Operating System Injection

A novel approach to help overcome the inherent issues with software-based approaches to memory acquisition was developed by Schatz [27] in their proof of concept tool called BodySnatcher. The approach injects an independent operating system into the kernel of the target machine. The target machine's native kernel execution is then frozen such that BodySnatcher can then provide an atomic snapshot of the machine's volatile data. A strength of the tool is that it does not matter if the target operating system has been subverted, as BodySnatcher only relies on its own tool set [27]. Although the technique shows promise as it produces atomic snapshots of memory, its availability is low due to a reliance on specific hardware platforms. In addition, the technique is limited to single processor execution [27]. As a result the acquisition of memory also takes a great deal of time.

3.3 Cold Booting Technique

Halderman et al. [34] present a novel approach to the acquisition of volatile memory. Their approach is based on the observation that information is not erased from volatile memory immediately after powering off a machine and may be recovered for a non-trivial amount of time [25]. Halderman et al. [34] present three methods for acquiring volatile memory contents, based on this phenomena of memory remanence. The first and most simple approach is to reboot the machine and launch a custom kernel with a small memory footprint that gives access to residual memory. The second is to briefly cut power, restore power and then launch the custom kernel. The second approach deprives the operating system of any opportunity to scrub the contents of RAM. The third attack involves cutting the power of the target machine and translating the RAM modules to a second computer which is configured to extract their state. This third approach denies the original Basic Input/Output System (BIOS) and computer hardware any chance to clear the memory.

[5] http://sourceforge.net/projects/mdd
[6] http://www.moonsols.com/windows-memory-toolkit
[7] http://www.mandiant.com/products/free_software/memoryze
[8] http://www.guidancesoftware.com
[9] http://www.gmgsystemsinc.com/knttools
[10] http://www.hbgary.com/fastdump-pro

In the approaches that involve the power to the system being cut, two methods have been developed to reduce the corruption of the remnant data [34]. The first is to cool the memory chips prior to removal, which improves data retention times substantially [34]. The second is to apply data correction algorithms. The level at which the cooling of RAM modules helps to reduce the decay of the information stored is significant. In the four machines tested, with no cooling the average amount of corruption after a few minutes was between 41% and 50%. In the cases where the RAM modules were cooled before restarting the machine and acquiring the contents, the amount of corruption after a few minutes was between 0.000036% and 0.18% [34]. This shows that this method is able to capture an atomic snapshot of RAM. Furthermore, the approach is readily available — the cooling of the memory chips was done with canned air and the memory capturing kernel was based on Syslinux[11].

A number of techniques have demonstrated the usability and validity of the Cold Boot approach. For example the proof of concept AfterLife, is able to copy the contents of RAM to an external storage medium after the target is rebooted [35].

3.4 Summary

This section has presented a number of memory acquisition techniques, both hardware-based approaches in Section 3.1 and software-based in Section 3.2. These methods of acquisition were discussed with regard to both their atomicity and availability. The hardware-based techniques that have been developed so far tend to be highly atomic, but require the preparation of a system prior to its investigation. As such these approaches are most viable for cases were business-critical servers need to be protected and have access to real-time information in the event of an intrusion. Conversely, software-based techniques for memory acquisition tend to be more available than hardware-based approaches, but are far less atomic in the production of memory images. There are some exceptions to this lack of atomicity however. In particular, virtualisation-based approaches are able to precisely copy the contents of a VM's volatile memory with little effort. Of course these approaches are limited to infrastructures that have been set up to use virtualisation. Another exception is images created by cold booting techniques. Such approaches are able to copy the contents of RAM with little data loss, using a custom kernel that is executed when a system is rebooted. An overview of the techniques with regards to their atomicity and availability is given in Table 4.

The current state of memory forensics acquisition techniques shows that there is no single best approach. Each of the approaches discussed have various advantages and disadvantages. In practice first responders will have to decide for each case whether it is worth capturing volatile data from the target system (such as if they observe encryption tools being executed) and then determine which acquisition method would be the best course of action. Section 4 examines the most common techniques within the memory forensics domain used to analyse the contents of a memory image that has been acquired by an investigator.

[11] http://www.syslinux.org/wiki/index.php/The_Syslinux_Project

Table 4: *Overview of Acquisition Techniques*

Technique	Atomicity	Availability	Comments
Dedicated Hardware (3.1.1)	High	Low	Atomicity may be compromised with hardware tampering.
Hardware Bus (3.1.2)	Moderate	High	Atomicity affected by random crashes in some cases.
Virtualisation (3.2.1)	High	High	In environments that utilise virtualisation.
Crash Dumps (3.2.2)	Low	Low	Not all memory is dumped, can overwrite system page file and requires registry modifications to work.
User-mode Applications (3.2.3)	Low	High	Easily subverted, will modify memory when capturing it and will not have access to entire memory range.
Kernel-mode Applications (3.2.4)	Low	Moderate	Easily subverted and will modify memory when capturing it.
Operating System Injection (3.2.5)	High	Low	Reliance on hardware platforms and very slow.
Cold Booting (3.3)	High	High	Requires off the shelf items and Syslinux

4 Memory Analysis

The physical memory of computers running Windows based operating systems contains a large amount of metadata necessary to manage the execution of processes running within the operating system [21]. This information can survive in volatile memory for several weeks, even if the system is under load [25]. As such the analysis of memory should be considered as an important component of a forensic investigation as it can provide a wealth of transient system state information not available in persistent storage. Various memory acquisition methods were discussed previously in Section 3. This section discusses the most common techniques used to analyse memory contents once they have been successfully extracted.

It is common practice for digital forensics practitioners to use command line tools such as `strings` and `grep` to search binary memory images for textual information such as user names, email addresses and passwords [7]. Analysts also use tools such as WinHex to identify headers or other suspicious data within memory images. Other tools such as `foremost` are also commonly used to "carve" out files from raw data. An analyst could use this tool to recover all of the images that reside in the memory – even those that are in memory addresses flagged as deleted that have not yet been overwritten. Methods such as these are easy to apply but create highly noisy data sets, require huge overhead and lead to a large number of false positives [36]. Beebe and Clark [36] argue that such tools require investigators to wade through thousands of search hits for even relatively small queries – most of which may be irrelevant to the their current objectives. Furthermore, even if interesting results are generated (for example `strings | grep "drugs"` finding matches within the raw data), they do not take into consideration the context of the result. For instance, if the string was found within the memory space of a user launched application (such as a web browser), it may be of probative value. However, if the same string was found in the memory space of malicious software it may not hold weight in court.

Recent research efforts focus on structured approaches to finding valuable traces in memory

images rather than manually searching images or using naïve approaches with strings-like tools [7]. Researchers have recognised that memory can be used to provide many vectors for analysis in an investigation and that there is a large amount of useful data stored in both the system and user address spaces, either directly in RAM or in the page file of the system [37]. New approaches tend to focus on determining what types of data are available in memory, how are these defined and where they are located in memory. The most common types of information that researchers are developing techniques to extract include:

- lists of currently executing processes and threads
- cryptographic keys
- system registry information
- established network connections and related network data such as Internet Protocol (IP) addresses and port information
- command history
- open files
- current users
- lists of rootkits or other malware.

Investigators can easily establish the state of a system with these types of information.

The remainder of this section examines these common analysis vectors and presents examples of novel approaches that have been implemented by researchers in the domain. It is structured as follows: Section 4.1 examines the extraction and analysis of process and thread data structures; Section 4.2 discusses the extraction of cryptographic keys from memory; Section 4.3 looks into the memory structures associated with the analysis of network connectivity; Section 4.4 discusses approaches used to identify open files; Section 4.5 discusses additional types of information that can be used to reconstruct the state of a system; and finally, Section 4.6 examines tools that have been developed that incorporate these analysis techniques.

4.1 Process and Thread Analysis

As discussed in Section 2, Windows operating systems maintain lists of EPROCESS and ETHREAD structures in memory, to help schedule and execute programs. Early attempts at extracting the state of a system are based on the concept of enumerating the lists of the processes and threads maintained in areas such as the Process Control Block (PCB) in a memory image [21]. These list-walking techniques are able to extract the list of currently executing processes and the threads attached to them from memory helping to provide a snapshot of the system from RAM. However, as these techniques enumerate through lists maintained by the kernel the results of the analysis may be subverted through techniques such as Direct Kernel Object Manipulation (DKOM). The FU-Rootkit for instance is able to unlink itself from the `ActiveProcessLinks` list (a member of the EPROCESS block) and thus avoid detection by methods that walk through lists and tables of processes in memory.

The limitations of enumeration style approaches to process and thread analysis have led to the development of signature-based scanning of memory. Schuster [21] developed a signature based scanning tool, that uses a set of rules to precisely describe the structure of a system process or thread. These signatures are used to extract all of the processes and threads from memory, even those that have unlinked themselves. As such, signature-based methods are useful in helping to detect malware that has hidden itself. This can be achieved by comparing the results of the system's task manager, or a traditional list-walking approach to the processes listed by a signature driven analysis. Any anomalies between the two could indicate the presence of malicious software [21].

Walters and Petroni [38] presented similar approaches, although they emphasised the need to use a signature that relies on components of a process or thread that are essential. By ensuring that only essential components of a process are used to identify processes and threads in memory images, these robust signatures can avoid attempts by malicious software to circumvent signature based scanners. For instance an attacker could set a non-essential field (such as the size of the block) to 0, which would cause Schuster's [21] method to fail. An attempt to set an essential field to an erroneous value to trick a robust signature would result in a guaranteed system-crash and thus cause the attack to be ineffectual [38].

Moving on from techniques that use hard-coded signatures to identify the structures of processes and threads in memory, Okolicia and Peterson [22] have developed a "Windows Operating System Agnostic" approach. This work extends existing analysis techniques in [21, 38]. [22]'s work is a novel example of how memory images can be analysed to extract information about the processes and threads executing on a system. The work presented aims to make it possible to take any arbitrary memory image from a Windows operating system and parse it without prior knowledge of the version. This removes the reliance on hard coded memory structures and signatures in order to search and analyse the memory image. Furthermore, the authors state that as new versions of Windows operating systems are released, the tool should continue to work.

The system developed in [22] first locates several debugging data structures in memory, such as `_DBGKD_DEBUG_DATA_HEADER`, `_KDDEBUGGER_DATA` or `_DBGKD_GET_VERSION`. These data structures are parsed to determine whether the image is from a 32-bit or 64-bit Windows operating system and whether PAE is turned on or off. Based on this information, the kernel page directory table is found. This table can be used to parse virtual addresses into physical addresses. Next, the base addresses of the kernel executable and of `tcpip.sys` are extracted from `_DBGKD_DEBUG_DATA_HEADER` directly and `PS_LOADED_MODULE_LIST` respectively. By examining the debug section of these two structures the authors tool is able to extract the Globally Unique Identifier (GUID) and age of the processes. With this information, the tool downloads Program Database (PDB) files from Microsoft's symbol server. These files contain a list of all of the defined symbols in an executable and their addresses. By parsing these files, the tool is able to export the kernel data structures from the executable. With these data structures, it is then possible to parse the memory image without any hard coded offsets, although the names of the EPROCESS and ETHREAD structures themselves are still hard wired into the tool.

Once the common operating system data structures have been identified and modelled (by extracting this information from the kernel's PDB file) they can be used to parse the memory image and extract process or system state information. For instance, the information contained in EPROCESS structures can be used to extract data about running processes. The PDB file of the process can be acquired in a similar fashion to that of the kernel executable, which enables the size and offset of the internal structures of the EPROCESS block to be examined. Using this

information a signature can be generated to help scan memory to extract all of the processes in the image. While signatures have been used previously (such as in [21, 38]) these signatures have always been generated prior to analysis, with specific versions Windows in mind. The novel aspects of the approach developed in [22] are that search signatures are generated dynamically, by pulling out pertinent version and platform information from the memory dump itself and then using this knowledge to acquire PDB files to aid in analysis.

4.2 Cryptographic Key Analysis

The analysis of memory images can be used to recover cryptographic keys which may provide access to encrypted information on a system. Casey et al. [10] argue that it is vitally important to capture volatile memory, as it may provide the only possible way to access devices that have FDE. Traditional methods involve "pulling the plug", which renders encryption keys in memory and any encrypted data inaccessible. Casey et al. [10] were able to recover a TrueCrypt password from a memory image. This password had been used to encrypt an entire disk volume. While their approach demonstrates the possibilities of capturing information to help decrypt data, the password was a known value.

Other approaches, such as those presented by Halderman et al. [34] show novel approaches to the recovery of encryption keys from memory, rather than known passwords. One of the methods they presented reconstructed keys from precomputed key schedules used by most encryption tools for efficiency. The usage scenario for this type of reconstruction was in the event that the original key had been damaged by memory decay, or during the acquisition process. Their methodology relies on the notion that most (if not all) disk encryption software pre-computes data from an original key, such as a key schedule for block ciphers or an extended private key for RSA. These precomputed values have much more structure than the original keys and can be used to reconstruct the original keys. The authors noted that in all of the disk encryption systems studied, precomputed key schedules were kept in memory for as long as the encrypted disk was mounted. Based on these schedules the authors were able to reconstruct Data Encryption Standard (DES), Advanced Encryption Standard (AES) and RSA private keys without resorting to brute force methods [34].

Another method from Halderman et al. [34] was aimed at pulling keys out of memory. The usage scenario for this approach is where the keys are not damaged and exist in full in the acquired memory image. Again, the authors target the key schedule instead of the key directly – searching for blocks of memory that satisfy the properties of a valid key schedule. The tool developed iterates through each byte of memory and treats each as the start of an AES key schedule. The tool checks the Hamming distance (i.e. the amount of substitutions required to transform one string into another) between each string in the potential key schedule and the strings that should be generated surrounding the key if it was real. If the substitution counts are low, the region must be close to a correct key schedule and the key is outputted. This was implemented for both 128-bit and 256-bit AES keys. The authors state that their approach was able to successfully recover keys from closed-source disk encryption programs without having to reverse engineer them. Their methodology successfully used memory to find keys and decrypted hard drives that were protected with BitLocker, TrueCrypt, dm-crypt and Loop-AES [34].

4.3 Network Analysis

Memory images are able to provide an insight into the network connectivity of a system. In addition to extracting process information from memory, work by Okolicia and Peterson [22] is able to extract the local and remote socket addresses of active network connections. Again their approach used Microsoft PDB files to identify the data structures and symbols used by an executable. In this case, the `tcpip.sys` executable was examined in memory, as the Windows kernel itself does not directly handle TCP/IP communications. Once the relevant symbols had been found the tool used the PDB file to locate them in physical memory and extract information from them. Through this method they were able to identify active TCP connections from a volatile memory image.

Network analysis provides insight into any malicious applications that may be executing on a system [7]. Such applications typically bind to pre-defined ports to allow attacks to execute arbitrary code on the target machine. Existing live forensics tools such as TCPView are able to reveal such threats [23]. Memory forensic analysis techniques can be used to help correlate data with these existing tools to gain a more thorough understanding of an incident.

4.4 Open File Analysis

In addition to identifying the processes executing on a system and related network activity, identifying the DLLs or files a process references may provide crucial evidence in an investigation. For instance, an attacker may have injected a malicious DLL in the address space of a legitimate process. These techniques commonly thwart traditional forensic analysis, as no data is written to a persistent source – the change is made entirely in RAM. By examining the PEB (see Section 2) of a process, analysis tools are able to extract the `Ldr` member. The `Ldr` pointer references multiple lists that contain the full name, size and base address of all loaded libraries. By iterating this list one may identify an injection attack, although this simple list walking technique is susceptible to manipulation.

An alternative approach to identifying files that a process has open, is to exploit the Virtual Address Descriptor (VAD) structure to uncover the files and objects a process has references to [7, 23]. The VAD data structure is maintained by the kernel to keep track of allocated memory ranges [23]. By examining the various components of the VAD structure, analysts are able to identify a control area that points to a number of file objects. These objects then contain the name of the file that is opened by the process. Open file analysis is an important aspect of any memory analysis, as it may be able to provide evidence that the user had opened a particular file for manipulation or viewing (such as an illegal image). Such evidence could be used to help refute common "Trojan" defence strategies that claim the defendant was not in control of the application or process that downloaded or opened a particular file.

4.5 System State Analysis

The most broad area of memory analysis involves extracting items from a memory image that can be used to reconstruct the state of a system. There are many different approaches used which target a large range of data structures that may provide insight into what a user was doing on the system before it was seized. Okolicia and Peterson [17] present a technique to extract the contents of the

Windows clipboard from RAM. The authors state that the clipboard is often overlooked as a source of digital evidence in forensic analysis, highlighting user passwords, copied sections of (potentially classified) documents and incriminating Uniform Resource Locators (URLs) as some types of information that could be found within the clipboard. Initially the authors used an application such as Notepad within a VM environment to transfer known information into the clipboard. A VM memory snapshot was used to then analyse the memory image. Then using a tool based on earlier work they searched within the memory for processes that had opened the `user32.dll` library [22]. If a process does not have this DLL loaded, then it is not able to access the clipboard. Based on the PDB of the `user32.dll` file, the location of symbols pointing to the clipboard memory for that application are found and the information at that location is extracted.

In addition to process and thread extraction techniques developed in [22], the authors also examined the extraction of system configuration from the Windows registry entries that are stored in RAM. They use a signature developed in a similar way to that used for process extraction, to search for "hives" in memory. Specifically, they search for the `_CMHIVE` field signature of `0xbee0bee0`. While most registry hives are stored on the disk as well as being loaded into RAM, there are two volatile only hives that are never stored to disk. These hives are called Hardware and Registry. The authors were able to extract information from these hives using their techniques, providing information about the current state of the system that would have been destroyed if a traditional analysis approach was used.

Aljaedi et al. [13] highlight the large variety of information types able to be extracted from memory to aid investigators in understanding the state of a system and to reconstruct user activity prior to capture. Specifically, their work focuses on extracting Internet browsing history from a memory image. Their analysis methodology only used simple approaches (such as searching for strings) but they were able to present some interesting findings. The authors launched a web browser and navigated to an online transaction store, which was secured with Secure Sockets Layer (SSL). They also used WireShark to capture traffic between between the browser and the server. The packets captured helped to determine the parameters of the session cookies sent from the web browser. These known values were then used to determine if session cookies are easily accessible in memory images, even after a web browser has been closed. In their experiments the cookies were identified in plain-text in RAM. Furthermore, the values entered into the forms on the secure website were also accessible in memory. For example, credit card numbers, Card Code Verification (CCV) numbers and the card holder's full name exist in memory even if the browser was in "private browsing" mode while the transaction took place.

4.6 Analysis Tools

Since the DFRWS in 2005 several tools and frameworks have been developed to help investigators with the analysis of volatile memory images. These tools have been developed in response to the level of difficulty and complexity in the analysis of memory images. There are a variety of products available for the analysis of volatile memory, including freely available open-source software and proprietary commercial products. The open-source frameworks tend to be more research focused and the commercial tools focused on delivering a product to digital forensic investigators. Some common tools and frameworks include:

- Windows Memory Toolkit from MoonSols[12]
- Responder (Professional and Community) from HB Gary[13]
- Memoryze from Mandiant [14]
- The Volatility Framework from Volatile Systems[15].

The Windows Memory Toolkit is a closed source, proprietary tool developed by MoonSols. It comes in both free and pro editions, with the primary difference being that the pro edition is scriptable and has interactive analysis modes. The tool supports the analysis of memory images acquired from a wide range of Windows operating systems, including Windows XP through 7. Unfortunately, neither version of the tool supports extensibility through plugins or an API and the analysis methods are not disclosed (due to the closed source nature of the product).

HB Gary's Responder (Professional and Community) provides physical memory analysis and malware analysis bundled into one application. It is able to extract information regarding the operating system, running processes, open files, network activity, open registry keys, passwords, web mail and malware. This tool utilises another HB Gary tool, Fast Dump Pro, to freeze the state of the operating system and extract the RAM. The professional edition is not free and the community edition is provided as an evaluation only. The source is closed and there are no methods of extending the software.

Memoryze from Mandiant is freeware digital forensic software designed to help investigators uncover malware and other malicious activity in live memory captures. The tool is able to perform acquisition and analysis, with support for full system memory acquisition and extraction of a single process's memory space to disk. The analysis techniques provided facilitate enumeration of running processes including those hidden by rootkits. For each running process the tool is able to identify open files, open registry keys, virtual address space, loaded DLLs, network sockets and active connections belonging to the particular process. The tool supports raw binary memory images captured via any means. Memoryze supports many Windows variants, including Windows 2000, XP (SP2/3), Server 2003 (SP1/2), 7 and Server 2008. The tool is available free of charge, but is not open source and does not provide any mechanisms to extend the capabilities of the product (through plugin interface or API).

The Volatility Framework is an open source software framework built to facilitate volatile memory forensics. A major advantage of the Volatility Framework when compared to other common tools is extensibility via a Python-based plugin framework. As a result of its plugin-based design Volatility supports a wide variety of input formats, including both Windows and Linux address spaces. Volatility provides analysis support in many ways, including extracting the image date and time, running processes, network connections, network sockets, open DLLs, open files, registry keys, process memory spaces and currently loaded kernel drivers. Furthermore, through community plugins the tool supports malware and rootkit detection. A full list of developed plugins is available at https://code.google.com/p/volatility/wiki/FeaturesByPlugin.

[12]http://www.moonsols.com/windows-memory-toolkit/
[13]http://www.hbgary.com/responder-pro-2
[14]http://www.mandiant.com/products/free_software/memoryze
[15]http://www.volatilesystems.com/

4.7 Summary

The analysis of volatile memory images should be considered an important component of any digital forensic investigation, in particular by first responders. Many techniques have been developed that enable crucial state information to be extracted from a memory image. Such techniques include the ability to list the currently executing processes (including those hidden by malware) and to identify files, registry keys and network connections that a process has open. By examining these files or network connections, theories regarding the users actions can be supported or refuted. For example, it can be determined whether it was the user or malware that had uploaded a confidential file.

One aspect of volatile memory analysis that could prove highly valuable to investigators is the ability to identify and extract encryption keys from memory images. In cases where systems have been hibernated, locked, or suspended and a suspect has employed FDE to protect illegal data, memory forensics may provide the only mechanism for the acquisition of that data. Brute force attempts to crack passwords or keys for encrypted containers typically fail to produce results in a meaningful time frame.

The most common analysis techniques have been incorporated into software tools and frameworks that can be used by investigators. These tools are able to streamline the process of acquiring and analysing the volatile memory.

5 Conclusion

Digital forensics is a branch of forensic science that involves the analysis and acquisition of data stored on digital devices. Specifically, it investigates the illegal or inappropriate use of such technology. Digital forensic investigations typically focus on the analysis of persistent data sources, such as the information contained on a hard disk or DVD. The process of investigation often involves removing power from the system, creating an exact copy of the static media and finally performing an analysis postmortem. However, these common practices fail to recognise the wealth of transient system-state information that is available in RAM.

RAM images can be used to reconstruct the current state of a system. Memory forensic analysis techniques search for any information that exists in RAM that can provide insight into the state of a system or corroborate any existing evidence in an investigation. Such information includes encryption keys, passwords, active network connections, open files and currently executing processes (including hidden processes and malware).

The most difficult component of memory forensics is typically the acquisition of a memory image in both an atomic and available manner. There are two broad genres of acquisition techniques, software-based and hardware-based. Software-based techniques (such as kernel drivers or user applications) are generally considered to be more “highly available”, in that they are more likely to function without prior setup or knowledge of the environment. Unfortunately, software-based acquisition methods require the allocation of memory on the target system in order to function. As such, approaches based on these techniques will always modify RAM as a by-product of capturing it. Furthermore, as software-based approaches rely on the underlying operating system they are susceptible to having their results modified during capture. In contrast, hardware-based approaches (such as dedicated hardware devices like Tribble and hardware-bus interfaces like FireWire or PCI) produce more accurate snapshots of memory, as they do not rely on the operating system and can

operate without requiring allocation of memory. Unfortunately, hardware-based approaches often rely on knowledge of the target environment and involve setting up the devices prior to an attack or investigation. As such, hardware-based approaches to acquisition have limited availability in most cases.

While there appears to be no single solution to the issues surrounding the acquisition of volatile memory, there are some novel approaches being developed. One such technique involves injecting a memory capturing operating system into the kernel of the target system. This approach is able to pause the target system and acquire RAM in a non-destructive manner. Another novel technique involves rebooting machines and executing a custom kernel to acquire memory on startup. This approach relies on the fact that data remains in RAM for a short while, even after a cold (power on/off) or warm (reboot) restart. Other approaches, such as the acquisition of RAM in virtualised environments, do not suffer from atomicity or availability issues. It is common for VM hypervisors to provide easy methods to capture a snapshot or image of the current state of a VM's operating system. These snapshots can, if the user desires, also capture the entire contents of memory in a file for later forensic analysis. This makes virtualised environments highly useful for testing and developing memory analysis techniques.

After a memory image has been acquired, many methods have been developed in order to analyse the information contained within it. Early methods enumerated the lists maintained by the Windows kernel to extract information about the processes running on a system. However, these techniques commonly failed to identify hidden processes or those that had maliciously unlinked themselves from kernel management lists. More advanced analysis approaches incorporate the use of process and thread signatures rather than simply enumerating lists. Such techniques are able to more reliably extract information about the state of the system, even in the case where malware has hidden itself. In addition to process extraction, other research has focused on extracting encryption keys, passwords, open files, registry keys and a wealth of other transient system state data. Of particular importance is the extraction of data such as pre-computed encryption key schedules, which only exist in RAM and may be used to unlock encrypted containers on a suspect's system.

Due to the high complexity of the in-memory data structures contained in memory images, several commercial and freely available analysis tools and software frameworks have been developed. The software generally includes common analysis techniques developed in the domain, such as process extraction, password recovery, open file analysis and so on. Of particular interest from a research perspective is the Volatility Framework. This software framework provides a highly extensible plugin-based architecture allowing researchers to develop new analysis modules or format readers. It has a healthy community, that provides many plugins to extract important data from memory images, including malware and rootkit analysis modules. Frameworks such as Volatility enable researchers and investigators to contribute new knowledge to the domain and easily share it by developing new plugins for the framework.

References

1. Casey, E. (2011) *Digital Evidence and Computer Crime: Forensic Science, Computers and the Internet*, third edn, Academic Press.

2. Mukasey, M. B., Sedgwick, J. L. & Hagy, D. W. (2008) Electronic crime scene investigation: A guide for first responders, second edition, *Innovation* **2011**(2), 93. URL – `http://www.ojp.usdoj.gov/nij/pubs-sum/219941.htm`.

3. National Institute of Justice (2009) *Electronic Crime Scene Investigation: An On-the-scene Reference for First Responders*, U.S Department of Justice.

4. National Institute of Standards and Technology (2009) National Institute of Standards and Technology: The CFReDS project, http://www.cfreds.nist.gov/. Accessed: 16/02/2012.

5. Shipley, T. G. & Reeve, H. R. (2006) *Collecting Evidence from a Running Computer: A Technical and Legal Primer for the Justice Community*, Technical report.

6. Farmer, D. & Venema, W. (2005) *Forensic Discovery*, Addison-Wesley professional computing series, Addison-Wesley.

7. Vömel, S. & Freiling, F. C. (2011) A survey of main memory acquisition and analysis techniques for the windows operating system, *Digital Investigation* **8**(1), 3–22.

8. Mrdovic, S., Huseinovic, A. & Zajko, E. (2009) Combining static and live digital forensic analysis in virtual environment, *in Information, Communication and Automation Technologies, 2009. ICAT 2009. XXII International Symposium on*, pp. 1 –6.

9. Riley, J., Dampier, D. & Vaughn, R. (2008) Time analysis of hard drive imaging tools, *in Advances in Digital Forensics IV*, Vol. 285 of *IFIP International Federation for Information Processing*, Springer Boston, pp. 335–344.

10. Casey, E., Fellowsb, G., Geigerc, M. & Stellatos, G. (2011) The growing impact of full disk encryption on digital forensics, *Digital Investigation* **8**(2), 129–134.

11. Lim, K.-S., Savoldi, A., Lee, C. & Lee, S. (2012) On-the-spot digital investigation by means of ldfs: Live data forensic system, *Mathematical and Computer Modelling* **55**(1–2), 223 – 240.

12. Sharif, M., Lanzi, A., Giffin, J. & Lee, W. (2009) Automatic reverse engineering of malware emulators, *in Security and Privacy, 2009 30th IEEE Symposium on*, pp. 94 –109.

13. Aljaedi, A., Lindskog, D., Zavarsky, P., Ruhl, R. & Almari, F. (2011) Comparative analysis of volatile memory forensics: Live response vs. memory imaging, *in Privacy, Security, Risk and Trust (PASSAT), 2011 IEEE Third International Conference on and 2011 IEEE Third International Conference on Social Computing (SocialCom)*, pp. 1253 –1258.

14. Burdach, M. (2006) Finding digital evidence in physical memory, *in Black Hat Federal Conference*, Sheraton Crystal City. Washington DC.

15. XForce ISS (2003) SQL Slammer Worm Propagation, http://xforce.iss.net/xforce/xfdb/11153. Accessed: 23/02/2012.

16. Carrier, B. D. & Grand, J. (2004) A hardware-based memory acquisition procedure for digital investigations, *Digital Investigation* **1**(1), 50 – 60. URL – `http://www.sciencedirect.com/science/article/pii/S1742287603000021`.

17. Okolicia, J. & Peterson, G. L. (2011) Extracting the windows clipboard from physical memory, *Digital Investigation* **8**(SUPPL.), S118–S124.

18. Zhang, L., Zhang, D. & Wang, L. (2010) Live digital forensics in a virtual machine, *in Computer Application and System Modeling (ICCASM), 2010 International Conference on*, Vol. 4, pp. V4–328 –V4–332.

19. Betz, C. (2005) memparser, http://www.dfrws.org/2005/challenge/memparser.shtml. Accessed: 27/02/2012.

20. Garner, G. M. (2005) kntlist, http://www.dfrws.org/2005/challenge/kntlist.shtml. Accessed: 27/02/2012.

21. Schuster, A. (2006) Searching for processes and threads in microsoft windows memory dumps, *Digital Investigation* **3, Supplement**(0), 10 – 16. The Proceedings of the 6th Annual Digital Forensic Research Workshop (DFRWS '06).

22. Okolicia, J. & Peterson, G. L. (2010) Windows operating systems agnostic memory analysis, *Digital Investigation* **7, Supplement**(0), S48 – S56. The Proceedings of the Tenth Annual DFRWS Conference.

23. Russinovich, M. E., Solomon, D. A. & Ionescu, A. (2009) *Windows Internals*, fifth edn, Microsoft Press.

24. Intel Corporation (2011) Intel 64 and IA-32 Architectures Software Developer's Manual, http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-vol-3a-3b_system-programming-manual.html. Accessed: 28/02/2012.

25. Chow, J., Pfaff, B., Garfinkel, T. & Rosenblum, M. (2005) Shredding your garbage: reducing data lifetime through secure deallocation, *in Proceedings of the 14th conference on USENIX Security Symposium - Volume 14*, USENIX Association, Berkeley, CA, USA, pp. 22–22. URL – `http://dl.acm.org/citation.cfm?id=1251398.1251420`.

26. Microsoft (2012) EPROCESS, http://msdn.microsoft.com/en-us/library/windows/hardware/. Accessed: 06/03/2012.

27. Schatz, B. (2007) Bodysnatcher: Towards reliable volatile memory acquisition by software, *Digital Investigation* **4, Supplement**(0), 126 – 134. URL – `http://www.sciencedirect.com/science/article/pii/S1742287607000497`.

28. Anderson, M., North, C. & Yiu, K. (2012) *Towards countering the rise of the Silicon Trojan*, Defence Science and Technology Organisation, DSTO-TR-2220.

29. Becher, M., Dornseif, M. & Klein, C. N. (2005) Firewire - all your memory are belong to us, *in CanSecWest Applied Security Conference*.

30. Boileau, A. (2006) Hit by a bus: Physical access attacks with firewire, *in RuxCon*.

31. Vidstrom, A. (2006) Memory dumping over FireWire - UMA issues, http://ntsecurity.nu/onmymind/2006/2006-09-02.html. Accessed: 14/03/2012.

32. Smith, J. E. & Nair, R. (2005) The architecture of virtual machines, *Computer* **38**(5), 32 – 38.

33. Vidstrom, A. (2002) PMDump, http://ntsecurity.nu/toolbox/pmdump/. Accessed: 16/03/2012.

34. Halderman, J. A., Schoen, S. D., Heninger, N., Clarkson, W., Paul, W., Calandrino, J. A., Feldman, A. J., Appelbaum, J. & Felten, E. W. (2009) Lest we remember: cold-boot attacks on encryption keys, *Commun. ACM* **52**(5), 91–98. URL – http://doi.acm.org/10.1145/1506409.1506429.

35. Vidas, T. (2010) Volatile memory acquisition via warm boot memory survivability, *in 2010 43rd Hawaii International Conference on System Sciences (HICSS)*, pp. 1 –6.

36. Beebe, N. L. & Clark, J. G. (2007) Digital forensic text string searching: Improving information retrieval effectiveness by thematically clustering search results, *Digital Investigation* **4, Supplement**(0), 49 – 54. URL – http://www.sciencedirect.com/science/article/pii/S1742287607000412.

37. Hoglund, G. (2008) The value of physical memory for incident response, http://www.google.com.au/url?sa=t&rct=j&q=hoglund%20g%20the%20value%20of%20physical%20memory%20for%20incident%20response&source=web&cd=1&ved=0CCUQFjAA&url=http%3A%2F%2Fwww.hbgary.com%2Fattachments%2Fthe-value-of-physical-memory-for-incident-response.pdf&ei=tipxT--YFo3NmAXGl9HGDw&usg=AFQjCNFJ2DqhQEm3sq8W8UIvoechkPrbMw&cad=rja. Accessed: 27/03/2012.

38. Walters, A. & Petroni, N. L. (2007) Volatools: Integrating volatile memory forensics into the digital investigation process, *in Black Hat DC*.

THIS PAGE IS INTENTIONALLY BLANK

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION DOCUMENT CONTROL DATA	1. CAVEAT/PRIVACY MARKING

2. TITLE	3. SECURITY CLASSIFICATION
Memory Forensics: Review of Acquisition and Analysis Techniques	Document (U) Title (U) Abstract (U)
4. AUTHOR	5. CORPORATE AUTHOR
Grant Osborne	Defence Science and Technology Organisation PO Box 1500 Edinburgh, South Australia 5111, Australia

6a. DSTO NUMBER	6b. AR NUMBER	6c. TYPE OF REPORT	7. DOCUMENT DATE
DSTO–GD–0770	015-765	General Document	November, 2013

8. FILE NUMBER	9. TASK NUMBER	10. TASK SPONSOR	11. No. OF PAGES	12. No. OF REFS
2012/1124100/1	07/348	CIOG	25	38

13. URL OF ELECTRONIC VERSION	14. RELEASE AUTHORITY
`http://www.dsto.defence.gov.au/publications/scientific.php`	Chief, Cyber and Electronic Warfare Division

15. SECONDARY RELEASE STATEMENT OF THIS DOCUMENT

Approved for Public Release

OVERSEAS ENQUIRIES OUTSIDE STATED LIMITATIONS SHOULD BE REFERRED THROUGH DOCUMENT EXCHANGE, PO BOX 1500, EDINBURGH, SOUTH AUSTRALIA 5111

16. DELIBERATE ANNOUNCEMENT

No Limitations

17. CITATION IN OTHER DOCUMENTS

No Limitations

18. DSTO RESEARCH LIBRARY THESAURUS

Computer Security
Memory
Computer Hardware

19. ABSTRACT

This document presents an overview of the most common memory forensics techniques used in the acquisition and analysis of a system's volatile memory. Memory forensics rose from obscurity in 2005 in response to a challenge issued by the Digital Forensics Research Workshop (DFRWS). Since then, investigators and researchers alike have begun to recognise the important role that memory forensics can play in a robust investigation. Volatile memory, or Random Access Memory (RAM), contains a wealth of information regarding the current state of a device. Memory forensics techniques examine RAM to extract information such as passwords, encryption keys, network activity, open files and the set of processes and threads currently running within an operating system. This information can help investigators reconstruct the events surrounding criminal use of technology or computer security incidents.