DATA ACQUISITION FROM VOLATILE MEMORY: A MEMORY ACQUISITION TOOL FOR MICROSOFT WINDOWS VISTA

by

Cheong Choong Wee Vincent

December 2008

Thesis Advisor: Timothy M. Vidas
Second Reader: George W. Dinolt

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** December 2008	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE Data Acquisition from Volatile Memory: A Memory Acquisition Tool for Microsoft Windows Vista			**5. FUNDING NUMBERS**
6. AUTHOR(S) Cheong Choong Wee Vincent			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

The focus of this research is on extracting data from the volatile random access memory (RAM) on a personal computer running Microsoft's Windows Vista operating system, while minimally affecting the existing data. The projected work includes the development of a kernel-mode device driver with the capabilities on one or more versions of Microsoft Windows Vista, a user-mode application that interacts with the driver, usage documentation and outcome of the research.

The main objectives of the research is to show the possibility of extracting information from the random access memory using a user mode application (with a suitable driver already installed) and to document the process of Window Vista driver development, so that future works in this area can benefit by putting more effort into specific research rather than configuring a development environment.

14. SUBJECT TERMS Windows Vista Memory Acquisition, Computer Forensics, Device Driver			**15. NUMBER OF PAGES** 127
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

THIS PAGE INTENTIONALLY LEFT BLANK

DATA ACQUISITION FROM VOLATILE MEMORY: A MEMORY ACQUISITION TOOL FOR MICROSOFT WINDOWS VISTA

Choong Wee Vincent Cheong
Civilian, Singapore Technologies Electronics Limited
B.Eng (Hon), National University of Singapore, 2003

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 2008

Author:	Choong Wee Vincent Cheong
Approved by:	Timothy M. Vidas Thesis Advisor
	George W. Dinolt Second Reader
	Peter J. Denning Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The focus of this research is on extracting data from the volatile random access memory (RAM) on a personal computer running Microsoft's Windows Vista operating system, while minimally affecting the existing data. The projected work includes the development of a kernel-mode device driver with the capabilities on one or more versions of Microsoft Windows Vista, a user-mode application that interacts with the driver, usage documentation and outcome of the research.

The main objectives of the research is to show the possibility of extracting information from the random access memory using a user mode application (with a suitable driver already installed) and to document the process of Window Vista driver development, so that future works in this area can benefit by putting more effort into specific research rather than configuring a development environment.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

The completion of this thesis is a culmination of my wonderful experience at the Naval Postgraduate School, Monterey. The results presented in this thesis also marks a major achievement for me and I would like to take this opportunity to express my heartfelt gratitude to the people that have helped me along the way and made achieving this feat a possibility.

I would like to thank my thesis advisor, Timothy M. Vidas, for his patience and guidance. Without which the completion of this thesis would not be possible. I would also like to thank my second reader, George W. Dinolt, for agreeing to participate and signoff on my thesis work. My gratitude also goes out to all the lecturers and professors who have taught me during my brief spell in the Naval Postgraduate School.

I would also like to give special thanks to my Singapore Comrades that for their help and support.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND AND MOTIVATION

1. Primary Memory (Random Access Memory)

The Primary Memory or Random Access Memory (RAM) also referred to as main or real memory is typically considered to be a volatile resource. The data residing on the memory becomes less reliable as power is weaned from it. Data stored in RAM persists briefly if power is not refreshed [1]. Some factors that affect the persistence of data in memory [2] include:

- System Activity
- Main memory size
- Data Type and
- Operating System

Both user space applications and the operating system kernel[1] use this form of storage to store data and execution codes. Since all executions leave a trace on the primary memory, there is an abundance of in-memory data at any time. Inspecting a snapshot of memory can reveal or help reconstruct the state of a system when the memory snapshot[2] was taken. This includes not only current running processes but also terminated processes as the portions of memory used by them (the terminated processes) will not be purged when these portions are no longer required for current operation [3].

[1] User space is often referred to as “ring 3” or “privilege level 3.” Similarly, kernel space is often referred to as “ring0” or “privilege level 0.” Both are artifacts of Intel’s architectural protection mechanisms present in the CPU.

[2] For purposes of this paper, a “memory snapshot” is simply an archive of the state of RAM at a point in time, saved to a file for example.

2. Analysis of Forensic Data from Volatile System Memory

It is therefore beneficial to analyze the content of RAM to gain insight into either the well being of the system or to gain access to information that is hidden or not available in plain sight. This information include open TCP/UDP ports/raw sockets/active connection, memory mapped files and caches such as web addresses and typed commands. Information that is usually obscured or not even present on disk, such as passwords, can potentially be retrieved from an analysis of RAM.

Since most modern operating systems do not purge the information on RAM until the space is needed, information related to recently terminated processes can also be extracted. This presents special significance for forensics purposes. The applications are countless and can range from trying to extract messages from an Instant Messenger of a kidnapped child to help with an investigation in trying to prove that someone has executed certain programs that he or she has recently terminated.

The applications, however, are not limited to just aiding criminal prosecution and investigations. Computer systems in large corporations and institutes such the U.S. Department of Defense are high value subversion targets. The sensitivity and value of the information on these systems makes it important not only to protect these systems but also to detect the presence of malicious software. An offline analysis of a memory image is one of the best ways to defeat most known anti-forensic technique and discovering Rootkits [2] [4].

3. Tools used for Volatile Memory Acquisition

There are many known techniques and tools available for capturing a volatile memory image. They can be broadly categorized into Hardware and Software based techniques, each method with its own suitability and limitations. These techniques are discussed below:

a. Hardware Based Techniques

The most direct method would be to have a piece of dedicated hardware for the sole purpose of creating a system memory dump, perhaps at the push of a button. A proposed method was to use a PCI card that disables the CPU and use DMA to access the host memory, creating the memory image on a connected device [5]. This has the major disadvantages of requiring the device to be installed prior to an incident and the cost associated with purchasing these devices for all computers in an organization.

Another technique makes use of the bus mastering capabilities of the Firewire protocol for imaging the memory of arbitrary host [6]. This is a feature of the Firewire protocol. Anyone with physical access to a machine with a working Firewire port can read and write to the main memory of the system. This technique obviously requires the target system to have a firewire port present. Making use of this technique would require that the target system be equipped with the necessary Firewire hardware and software (device driver). More importantly, the same vulnerability that is exploited for imaging the memory can also be used to subvert the system. This is therefore not a good way for acquiring volatile memory.

b. Software Based Techniques

A common UNIX program that is used for low level copying and conversion of data is dd. Specifying the /dev/mem as the input file parameter, dd can be used to image the data in the RAM of UNIX systems. George Garner successfully ported this tool to Windows system in 2006 [7].

Helix is a bootable CD with a customized distribution of Ubuntu Linux [8]. It is a computer forensic toolkit used for incident response. It is also capable of creating an image of the data that exists in RAM. This feature however does not work in the newer versions of the Windows Operating System as user mode applications are no longer capable of directly accessing RAM. As such this is not a viable way to acquire a memory image of system running Windows Vista.

Some implementations of Virtual Machine Concept [9] such as VMWare [10] provide a means of taking a snapshot of the state of a machine. This includes the CPU, memory (main and virtual), hard disks, active processes, etc. When a snapshot of the machines is taken, the contents of the memory is preserved and stored as a file on the host machine[3]. This form of capturing the image content of memory is particularly useful for research and analysis of memory images. However, this method does not address the issue of creating a memory image on a live (non-virtualized) system.

4. Windows Operating System

The focus of this research is on extracting data from the volatile random access memory (RAM) on a personal computer running Microsoft's Windows Vista operating system, while minimally affecting the existing data. The research work includes the development of a kernel-mode device driver with the capabilities on one or more versions of Microsoft Windows Vista, a user-mode application that interacts with the driver, and a thesis that documents the development process and the research findings.

The device driver is capable of exposing the volatile memory as a read only media (similar to a CD-ROM). The MemReader tool will also include a user mode application that makes use of this device driver and writes the content of the memory unto a specified storage media.

The thesis also documents the developmental process and captures information on kernel driver development. This includes some problems encountered and certain knowledge and *gotchas* that will aid future students working in this area of research.

[3] Often RAM is preserved as a single file with a .vmem extension.

B. RESEARCH METHODOLOGIES AND ORGANIZATION

1. Primary Research Question

- Is it possible to build a driver capable of extracting data from the volatile memory? If so, in what ways will the machine state prior to acquisition (volatile and non-volatile memory) be affected by this method of acquisition?

2. Subsidiary Research Questions

- What are the different ways the device driver can be employed (i.e. at runtime, boot time) to extract data and what are the differences (between these ways)?
- How portable is the device driver between different Windows Vista versions?
- How does the tool differ from the plethora of tools becoming available in the market[4] in obtaining an image of volatile RAM and how does the image captured by our tool compare with those captured by the others?

3. Assumptions

Memory management is often carried out dynamically by the operating system to cater to the needs of existing and ongoing services and processes. This also means that the content of the memory is constantly changed; hence, the term “volatile memory”. It is also relatively difficult to determine the exact content and location of data residing in memory. This makes verifying the output of any MemReader a difficult job.

[4] ManTech’s DD was not available when work on this thesis was started. Below we provide a comparison of our work with “DD.”

The content of RAM can also be captured when a snapshot of a system is taken by VMware. To use this method of comparison for this research, the snapshot would need to be taken immediately after the image of RAM is created by an imaging tool. The instability caused to systems during device driver development means that it is not always possible for the tool being developed to terminate gracefully before it crashes the system. This method of collecting the RAM images for comparison was therefore not chosen.

For the purpose of this research, the ManTech's DD (MDD) tool was used to create a RAM image that was used as a baseline for comparison with the RAM image created by the MemReader driver and application. The RAM image create by MDD is assumed to be correct and therefore suitable for validation purposes.

4. Scope

Windows Vista Business was the version of the Windows operating system used for both the development and testing of the tool. Although the Window Driver Kit (WDK) [39] provides the development environment for some versions and architectures (like the x86 and x64). This project was only interested in the development of a tool for the Intel x86 architecture.

Since version of Windows later than Windows Vista require kernel mode drivers to be signed, a self-signed certificate created using the MakeCert tool in WDK can be used for self signing purposes.

5. Chapter and Appendix Overview

This thesis is comprised of the following chapters and appendices:

Chapter I: Introduction – This chapter gives an overview on the thesis. This includes some background information, objectives and scope of the research work and a brief overview of the proceeding chapters.

Chapter II: Background of Windows Device Drivers – This chapter touches on the background of Windows Device Drivers that will be beneficial for understanding the rest of the thesis. The information is also a good starting point for anyone that is interested in taking up Windows device driver development.

Chapter III: Requirements and High Level Design – The chapter describes the requirements for developing a device driver for Windows. This includes the testing the debugging requirement. There is also a comparison between the available tools that can extract a RAM image on a Windows system and the driver that was developed.

Chapter IV: Implementation – This chapter is presents the implementation strategy adopted in this research and outlines some of the problems encountered.

Chapter V: Test Methodology – This chapter presents the methods adopted in this thesis to verify the correctness of the work.

Chapter VI: Results and Analysis – This chapter presents the results of the testing. This includes method used in gathering the results and an analysis of the collected images.

Chapter VII: Future Work – This chapter proposes areas of research work that can be further extended from the outcome of this research.

Chapter VIII: Conclusions and Recommendations – This chapter summarizes the conclusion, discusses the major challenges and presents certain recommendation regarding the ramifications of this research.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND OF WINDOWS DEVICE DRIVERS

A. OVERVIEW OF THE SIGNING PROCESSES

Digitally signed kernel-mode software is a process that attempts to ensure security on computer systems. Microsoft's Windows Vista relies on digital signatures for kernel-mode code to increase the safety and stability of the Windows platform. All 64-bit kernel-mode software intended for x64-based computers running Windows Vista must be digitally signed. This applies to boot-start drivers for x86 and x64 versions of Windows. Microsoft however encourages publishers to digitally sign all software, including device drivers for both 32-bit and 64-bit platforms [35].

1. Kind/ Type of Signatures

a. Test-Signing

This refers to using a test certificate to sign a pre-released version of software for use on test computers. There is a MakeCert [40] utility program included in the Window Driver Kit (WDK) installation, which generates a self-signed certificate that developers can use to sign kernel mode binaries. The boot configuration of the machine must be changed, using the bcdedit.exe command [57], before test-signed drivers will be loaded.

2. Software Publishing Certificate (SPC)

A Software Publishing Certificate (SPC) is used for signing the catalog file for a driver package. A SPC can be obtained from a Commercial Certificate Authority (CA). The list of CAs that provides SPC that can be used for kernel-mode code signing is available at the Microsoft Website [41]. A corresponding cross certificate from Microsoft is needed together with the root CA issued by the SPC, so that the signature can be verified by a trusted root authority known to the Windows Vista operating system.

B. MINIMAL REQUIREMENTS TO GET THE DRIVER LOADED

A driver using the Kernel Mode Driver Framework (KMDF) provided by the WDK is a kernel module that lives permanently inside the system. In many ways, a driver is treated by Windows as a regular service that can be started and stopped just like any other system service. The two steps involved in installing a driver are:

1. Registering the driver as a system service
2. Enabling/ Starting the driver/ service

The Service Control Manager exposes APIs that can be used to install and start a device driver/ service from any win32 application. It is therefore possible to register and start a driver programmatically.

1. Registering Drivers and Services

A service can registered in one of the two ways:

a. CreateService API

The CreateService API can be used to register a driver as a system service. Details of the API call can be found at Microsoft Developer Network (MSDN) documentation [20]. The service StartType [18] can be specified, as a parameter to the CreateService API, when the service is being registered.

b. Manually Editing the Windows Registry

The CreateService API creates the relevant entries and values in the Windows registry. It is possible to manually create these values in the following registry by adding a new subkey with the name of your driver. The registry entry will look like:

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\<driver name>
```

The registry values of the name, path and start-type of the driver need to be supplied.

Registry Value	Description	Example
DisplayName	Name of service/ driver as it appears on the service list	IP Traffic Filter Driver
ImagePath	Full path to the driver (or just the filename if it resides in the system32\drivers directory)	system32\DRIVERS\ipfltdrv.sys
Start	The point where the service is started	3 (SERVICE_DEMAND_START)[5]
Type	The type of service	1 (SERVICE_KERNEL_DRIVER)[6]

Table 1. Example Service Registry Entry

2. Starting the Service

After the driver has been registered as a system service, it can be loaded (and unloaded) using the Service Control Manager. It can be started either programmatically using the StartService API or using the Windows net command:

```
net start <service name>
```

3. Using the Windows Control Panel

Using the Add Hardware option in the Windows Control Panel, a device driver can be added to the system. This will be described in more detail in the section on driver packaging in Chapter IV.

C. INSTALLATION FOR 32-BIT AND 64-BIT PLATFORM

A native driver package must be provided for installation on different platforms. A single 32-bit installer can however be used to install the driver package on the different platforms. This will be described in more details in the section on driver packaging in Chapter IV.

[5] Some other possible values include: SERVICE_BOOT_START (0), SERVICE_SYSTEM_START (1), SERVICE_AUTO_START (1), SERVICE_DISABLED(4) [37].

[6] Some other possible values include: SERVICE_FILE_SYSTEM_DRIVER (2), SERVICE_WIN32_OWN_PROCESS (8), SERVICE_WIN32_SHARE_PROCESS (9) [37].

D. OVERVIEW OF THE WINDOWS DRIVER

Device drivers mediate communication between the kernel and a device. The Microsoft’s Windows operating system provides an abstract device support interface called a driver model to facilitate this communication. Device drivers provide the implementation to this interface to support the specific requirements of the device. Device drivers operate much like services provided by the operating system; they handle communication between applications (or other drivers) and a device.

1. Kernel Mode Drivers

Kernel Mode Drivers run entirely in kernel space and are capable of communicating and transferring data directly to kernel mode components. This means that Kernel Mode Drivers can bypass the mediators lying between user and kernel space. Tasks involving kernel mode components therefore carry less overhead when carried out by a Kernel Mode Driver as compared to User Mode Drivers. Moving a driver into the kernel can also sometimes improve performance. The flipside of it all is that when a kernel mode driver fails, it can crash an entire system, whereas the failure of a user mode driver causes only the current process to crash.

2. Kernel Mode Driver Framework (KMDF)

The KMDF is an infrastructure for developing Kernel Mode Drivers. The framework is a skeletal device driver that can be customized for specific devices and event handling. The framework drivers are customized by setting object properties, registering callbacks of relevant events, and using and including features that are needed. Not only does KMDF define a set of object models, it controls the lifetime and the memory allocation of these objects.

a. KMDF Components

Table 1 lists the KMDF components that are installed part of WDF.

Component	**Location**	**Description**
Header files	wdf/inc	Header files required to build KMDF drivers
Libraries	wdf/lib	Libraries for x86, x64, and Intel Itanium architectures
Sample drivers	wdf/src	Sample drivers for numerous device types; most are ported from Windows Driver Development Kit (DDK) WDM samples
Tools	wdf/bin	Tools for testing, debugging, and installing drivers; includes the redistributable KMDF co-installer, WdfCoinstaller*nn*.dll
Debugging symbols	wdf/symbols	Public symbol database (.pdb) files for KMDF libraries and co-installer for checked and free builds
Tracing format files	wdf/tracing	Trace format files for the trace messages generated by KMDF libraries and co-installer

Table 2. KMDF Components

b. Structure of a KMDF Driver

A KMDF driver is identified by a *DriverEntry* function that represents the driver's primary entry point. Drivers that support Plug and Play devices also have An *EvtDriverDeviceAdd* callback. Specific I/O request events for a particu2lar queue can also be registered as *EvtIo** callbacks. A simple device will have the above-described functions and nothing more. KMDF also includes code to support power management, Plug and Play and Windows Management Instrumentations.

3. Description of the KMDF API

The Kernel Mode Driver Framework (KMDF) object model defines a set of objects that represent common driver constructs such as devices, memory, queues, I/O requests, and the driver itself. The framework objects have well-defined life cycles and contracts, and a WDF driver interacts with them through well-defined interfaces and they expose programming interfaces that handle their properties, methods that perform actions, and events handling through callback registration [36]. Listed below is the description of some of the prominent Application Programming Interfaces that were used in the development. This list is provided as a reference for those not familiar with KMDF when studying the source code that is attached as Appendix E.

1. NTSTATUS DriverEntry(IN PDRIVEROBJECT Driver Objects, IN PUNICODE_STRING RegistryPath) – This is the entry point for the application programming of a Windows device driver. It is the first function that will be called by the framework [46].

2. WDF_DRIVER_CONFIG_INIT() – This method initializes the WDF_DRIVER_CONFIG structure and the second parameter to the method also registers the user defined MemReaderEvtDeviceAdd function [47].

3. WdfDriverCreate() – This method creates a framework driver object for the calling driver. Specific driver attributes can also be defined in the user allocated WDF_OBJECT_ATTRIBUTES structure [48].

4. WdfDeviceCreate() – This method creates a framework device object. Similar to the DriverCreate method, specific device attributes can be defined. This method is called in the user defined MemReaderDeviceCreate function, which creates a device and the software resources needed by the MemReader driver [49].

5. WdfDeviceCreateDeviceInterface() – This method creates a device interface for a specified device, i.e. the device create with WdfDeviceCreate. The device interface is a symbolic link that applications can use to access the device [50].

6. WdfIoQueueCreate() – This method creates and configures an I/O queue for a specified device, i.e. the device create with WdfDeviceCreate. Specific queue attributes, such as user defined

queue callback events, can also be defined in the user allocated WDF_OBJECT_ATTRIBUTES structure [51].

7. WdfRequestRetrieveOutputMemory() – This method retrieves a handle to a framework memory object that represents an I/O request's output buffer. It is used to receive information that the driver provides to the originator of the request, such as a read request [52].

8. WdfMemoryCopyFromBuffer() – This method copies the contents of a specified source buffer, such as the framework memory object, into a specified memory object's buffer [53].

9. WdfRequestCompleteWithInformation() – This method stores completion information and then completes a specified I/O request with the supplied completed status, as the second input parameter, to the calling application [54].

10. WdfMemoryCopyToBuffer() – This method copies the contents of a specified memory object's buffer into a specified destination buffer. This is a reverse WdfMemoryCopyFromBuffer operation [55].

11. WdfRequestRetrieveInputBuffer() – This method retrieves the an I/O request's input buffer. It is used during a write operation to write to the where the application passes down information to the driver. This is a reverse WdfRequestRetrieveOutputBuffer operation [56].

A *MemReader* tag precedes all the user-defined functions that are not part of the API. These include user-defined callback, plug and play related and other functions.

THIS PAGE INTENTIONALLY LEFT BLANK

III. REQUIREMENTS AND HIGH LEVEL DESIGN

A. BUILDING A KERNEL DRIVER

The goal of this research is to build a Kernel Mode Driver Framework (KMDF) driver for the Windows Vista operating system that can interface with a user-mode application and extract data from the RAM. The discussion below will be limited to areas related to this research.

1. Minimal Requirements to Obtain Developmental Environment

The Windows Driver Foundation (WDF) defines a single driver model for creating object-oriented, event-driven drivers for either kernel mode or user mode. The Windows Drivers Kit (WDK), which contains the Driver Development Kit (DDK), is the primary resource needed for driver development using the WDF. The latest version of the WDK can be downloaded from Microsoft Connect [13]. A Windows Live ID or Microsoft passport is required to join the Microsoft Connect. Detail instructions on obtaining and installing the WDK can be found on the Windows Hardware Developer Central (WHDC) website [14].

a. Operating System Requirement

Any recent version of Windows can be used to develop and build KMDF drivers for Windows 2000 or later versions of Windows. The appropriate build environment configuration must be specified when using the build.exe utility to target a driver for a particular version and CPU architecture.

It is, however, recommended to install, test, and debug the driver on a system that is running the target version of Windows and hardware that is the same as or similar.

b. Hardware Requirements

It is highly recommended that the developmental work configuration includes a computer that supports some form of Symmetric Multiprocessing or Multithreading (such as hyperthreading). This allows certain type of bugs, such as race conditions, to be detected.

It is also highly recommended that the hardware and the Operating System is matched, i.e., a 64-bit computer that runs x64 version of Windows. Some critical errors can only be detected on a 64-bit system. For the purpose of this thesis, a 32-bit system was used.

Two computers are needed to carry out "live" debugging of KMDF drivers, one to host the debugger and the other to host the driver (and requisite operating system) to be debugged. Kernel-mode driver bugs commonly cause system crashes and can corrupt the file system, causing loss of data. It is therefore necessary to have the debugger and driver on separate computers.

2. Development Environment

a. Build Utility

The Build utility in the Windows Driver Kit (WDK) invokes the correct tools with the appropriate options to build your driver. The Build utility is a wrapper around the Microsoft NMAKE utility, so it handles issues like checking dependencies to ensure that the correct files are rebuilt after a change. Building the samples provided in the WDK is a good starting ground for familiarizing and understanding the Build utility.

b. Build Environment

The WDK has build environments for four versions of Windows namely:

- Windows 2000
- Windows Server 2003
- Windows Vista and Windows Server Longhorn and
- Windows XP

For each version of Windows, there is a checked and free build environment for each CPU type. Drivers compiled in the checked build environment will have many of the optimizations disabled and conditional code for debugging included. Production drivers are compiled in the free build environment will have debug code disable and will contain optimized code.

c. Editor and Integrated Development Environment (IDE)

The source code in this research was edited using the Windows wordpad.exe. Microsoft Visual Studio can however be used as both an editor and an IDE for driver development. It would require all Microsoft Visual Studio options to be configured to match those of WDK and it should be configured to invoke the build.exe utility (provided by WDK) using a batch command line. However, Microsoft Visual Studio was not used in this research. A more detailed discussion on configuring Microsoft Visual Studio for driver development can be found on the OSR Online [58] and Hollis Technology Solution [59] website.

3. Debugging Tools

a. Windows Debuggers

Microsoft provides a 32-bit and 64-bit set of Debugging Tools [22] for debugging the operating system, applications, services, and drivers that run on the operating system. Out of the four debuggers provided by the installation package, only Microsoft Kernel Debugger (KD) and Microsoft Windows Debugger (WindDbg) are capable of debugging kernel-mode programs and drivers.

The setup of kernel-mode debugging is an elaborate one that requires both hardware and software configuration. KD and WinDbg are both powerful debuggers that are capable of monitoring the behavior of the operating system itself and full source level debugging for Windows kernel, kernel-mode drivers and system services respectively.

b. Debug Output Viewer (DebugView)

DebugView [23] is an application for monitoring the debug outputs on your local system, or any computer on the network that is reachable via TCP/IP. It is capable of displaying both kernel mode and Win32 debug outputs that are generated by applications and device drivers. *DebugView* will capture the output of the following functions:

- Win32 OutputDebugString
- Kernel-mode DbgPrint
- All kernel-mode variants of DbgPrint implemented in Windows XP, Server 2003 and Vista

DebugView also extracts kernel-mode debug output generated before a crash if it was capturing at the time of the crash.

In Windows Vista, DbgPrint defaults to the DEFAULT debug component and the default settings for this component is to hide all outputs. To display the DbgPrint outputs, go to the registry path at HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SessionManager \ add a new key called “Debug Print Filter” and add the following value “DEFAULT” : REG_DWORD : 0xFFFFFFFF and then reboot.

DebugView is a simple tool that requires little setup before it is ready for use. It, however, requires elevated privileges to display kernel debug outputs. This tool is used extensively in the debugging during developmental work due to its simplicity and ease of use.

B. DESIGN COMPARISON TO SIMILAR TOOLS

Several free and commercial tools are capable of creating a memory image of the data contained in RAM. This includes the DD released by ManTech [16], Helix released by eFense [8] and the Forensics Acquisition Utilities [7] by George M Garner. Microsoft changed the way memory is accessed on Windows Vista and has moved the access from user space into kernel space. The three tools all require elevated privileges for them work to on Windows Vista.

The source code and design of the version 1.2 of the ManTech DD tool was studied in more detail for literary and comparison purposes. The package released in version 1.2 includes an executable application file and a device driver. Similarly, the goals of this development will include an executable application and a device driver. The application needs to be started in a command prompt that has elevated privileges.

Once the DD application starts, it determines if it is executed in the correct environment, such as operating system and architecture. It then proceeds to load the device driver that came with the package and starts it as a system service. The device driver opens a handle to the physical memory segment. This handle is passed to the application that utilizes it to create an image of the RAM. Since the handle to the physical memory section is being used in the application, it requires elevated administrator privileges.

One of the objectives of this research is to show that utilizing an appropriate device driver; a user space application can also be used to create an image of RAM. For the purpose of this research, a kernel mode device driver was developed and loaded into the kernel. The driver opens a handle to the physical memory segment and when the user mode application interacts with the device driver, the driver maps sections of the physical memory into the process's virtual address space. The driver then copies the information from the physical memory into the buffer of the application. The application then saves the data on the buffer into an image file in a filesystem present on the computer system. Figure 1 is a graphical representation of this process.

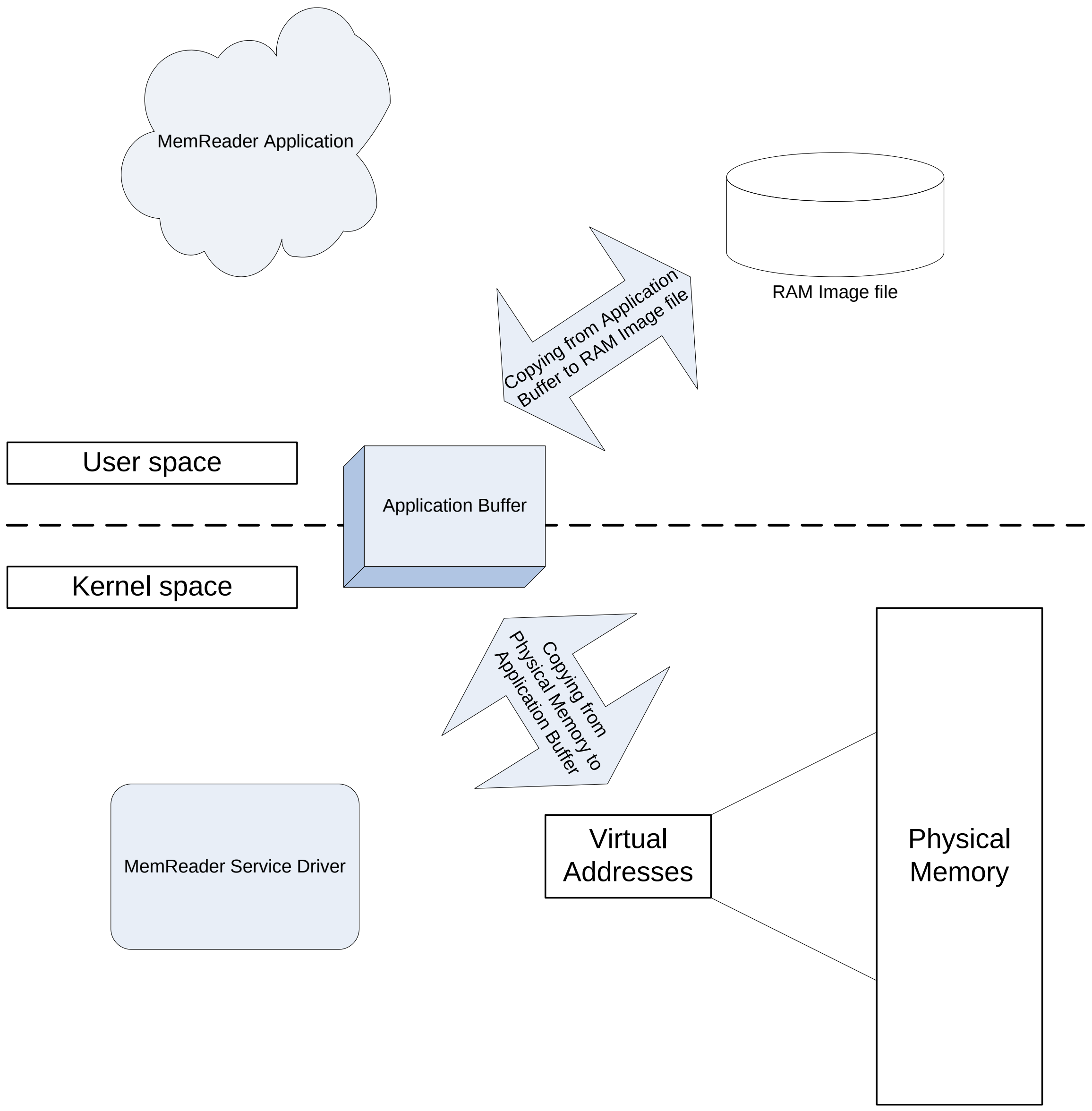


Figure 1. Graphical Representation of MemReader Application and Driver

IV. IMPLEMENTATION

A. IMPLEMENTATION OBJECTIVE REVIEW

Part of the objective of this research is to develop a tool that is capable of extracting the contents of the volatile RAM from a Windows Vista system. Since the proposal for this research was approved, ManTech International Corporate [15] has released a similar open source tool [16] for capturing memory on Windows Vista and Windows 2003 Server.

The tool release by ManTech consisted of a device driver and a separate Windows application. The device driver does not need to be installed on the system before the application is run. However, since a driver is being installed, administrative privileges are required for the user running the tool .

The purpose of the research is therefore to develop a driver that can be preinstalled on Windows Vista system such that a user mode application can be used to extract a copy of the memory when it is needed, providing organizations a method of incident preparation in addition to incident response.

Such a driver can be preinstalled on systems that are vulnerable to attacks, extracting an image of the RAM will have significant forensic benefits. An example would be honeypots that have been setup by system administrators.

B. IMPLEMENTATION STRATEGY

1. Windows Driver Kit (WDK) Samples

Besides the development environment, there is a wealth of useful information that is distributed together with the WDK. One such source is the samples that are included with each distribution. The different samples not only provide a starting point for understanding the User Mode Driver Framework

(UMDF) [17] and the Kernel Mode Driver Framework (KMDF) [18] but also the different object models and functionalities.

The sample codes are all “ready to compile” in the WDK development environment and no modification to the system or any files is required. Depending on the chosen type of build environment (checked or free) for compilation, the output files will be built accordingly. The built driver (<*sample*>.sys) be installed using a Co-installer that should come with the distribution of the WDK. This will be described in more detail in a later chapter.

The sample codes are therefore good starting points for developmental work to begin. This greatly reduces the learning curve of driver development and environment setup. The samples were studied and the one with the closest structure and functionality that fits the purpose of this research was selected to be the starting point of development work. Modifications were made to the sample code to fulfill the functionality requirement of this research.

2. The Echo Sample

The very nature of the driver requires access to the physical memory and some form of I/O between a kernel mode driver and a user mode application limited the workable set of samples to those that made use of the KMDF. The Echo sample, though not the only sample that fulfilled these two requirements, was chosen for its simplicity and elegance.

The driver in the echo sample demonstrates the use of a default I/O queue, it requests start events, cancellation events, and a synchronized Deferred Procedure Call (DPC). The modifications were mainly limited to how the driver interacted with the physical memory and how the data read from the physical memory is transferred onto the I/O queue. The full source code can found in Appendix C.

3. Development Process

a. Functional Description

The driver is an interface between the application and the physical memory device. As such, it has to be capable of accessing the data on the physical memory device and transferring it to the user mode application. The driver acts as an intermediary that transfers data from the physical memory to the buffer of the user mode application. The user mode application will need to write the data it receives from the driver into a file.

Before the driver can gain access to the data on the physical memory, it will need to get a handle to the physical memory device. This can be accomplished by the ZwOpenSection [24] function call to the \Device\PhysicalMemory [25] location. This handle is used by the driver during subsequent read events, EvtIoRead, where different sections of the physical memory section will be mapped into the virtual address space of the application that invoked the driver. The complete source code is attached as Appendix E.

b. Development Cycle

The DbgPrint function is used extensively in the driver code for verification and debugging. The driver is modified incrementally either by adding new functionalities such as retrieving a handle to the physical memory and by trimming the unnecessary functions such as the timer for Deferred Procedure Calls from the sample code.

As the driver is modified, the older version of the driver is uninstalled and a new version is installed on the target system and tested. Installation of driver is done through the Add Hardware menu in the Control Panel. It is worth noting that when a driver is uninstalled, through the Device Manager in the Control Panel, only the devnode associated with the device is removed. The device instance no longer exists but the device package is still in the driver store. When the driver code is modified and subsequently rebuilt, there will be an updated version of the .sys driver file. The information in the INF file is however unchanged and remains the same as prior compilations. When the newly built package is installed (Control Panel > Add Hardware), Setup simply re-associates the devnode with the previous version of the driver package in the driver store by recreating the appropriate registry entries.

To ensure that the updated version of the driver package is installed correctly, the driver package must be deleted from the driver store during uninstallation. This can be done by selecting the “Delete the driver software for this device” checkbox during the uninstallation process (Control Panel > System > Device Manager). The WDF co-installer DLL file must be in the same directory as the INF and driver (.sys) file when the updated driver is being installed.

Once the updated driver package has been installed, the debug outputs are captured using DebugView and the application debug output are printed to the console. Together with the information saved in the dump file (RAM image), the correctness of the driver and the application is verified. The cycle of modification, rebuilding, uninstallation, reinstallation and testing is continued until the driver and the application are verified to be working correctly.

C. DEBUGGING

Debugging a KMDF driver for beginners is a tricky business. Since kernel mode drivers execute in a lower and more privileged level of the operating system, they do not have straightforward access to the user mode components such as the standard output. Debug and print statements are therefore not as easily displayed as in user mode applications. The ramifications of any violation in kernel mode components are also much greater and they are almost certain to be met with a bug check (system crash). Operating in kernel space also means that the components will need to cater to operating features such as paging and interaction with other system drivers. A careless mistake can sometimes be propagated down the chain and be amplified into a big problem.

1. Printing Debug Statements

Printing output statements represents the simplest debugging aid. DebugView is therefore the most widely used debugging tool during the development. The DbgPrint API call be made from anywhere within the driver code and it will be displayed with drivers built in the check and free environments. The DbgPrint function [26] takes in a variable argument list much like its user mode counterpart – printf, and can be formatted to display variable values.

DebugView was used at all times during the testing and debugging phase. This ensured that debug statements could be retrieved later in the event of an unexpected bug check. To retrieve the debug statements prior to a bug check, use the process dump file from the File option in top menu bar and browse to and select the memory dump file created by the bug check.

2. Analyzing the Memory Dump File

When a bug check is issued, a crash dump file will be created proved that the system is configured to create it. To configure for a full memory dump during a bug check, go to the System Menu in the Control Panel (Hold Windows key + Pause/ Break). In the “Advance” tab in the “Advance System Settings” menu, select “Settings” for “Startup and Recovery”, choose a “Complete memory dump” in the “System failure” section. The memory dump file contains a snapshot of useful low-level information about the system that can be used to debug the root cause of the bug check.

WinDbg was used to analyzed the memory dump files when the MemReader driver caused any system crash. The stack traces and device driver that led to the crash can be inspected. The appropriate symbol package [44] need to be installed before trace information of the system drivers is observable.

D. DEVICE DRIVER PACKAGE

A driver package consists of all the hardware and software components that are needed to support the device under Windows [21]. Since the driver that was developed in this research is a pure functional driver not associated with any hardware, the driver package we create will only consist of software components. This includes the driver files and the installation files.

1. Driver Files (.sys file)

The driver files provide the I/O interface between the application and the physical memory. The driver files are usually in the form of a dynamically linked library with a *.sys* file extension. Upon installation of the driver, the *.sys* file is copied to the %windir%\system32\drivers\ directory.

2. Installation Files

These files are needed for driver installation. Drivers can be installed through the Add Hardware menu in the Control Panel.

a. Device Setup Information file (.inf file)

A .inf file contains information that the system Setup components use to install support for the device driver. The registry entry information related to the driver makes up part of the .inf file. The .inf file of the driver package for this thesis is attached as Appendix E.

b. Co-installers

A co-installer is a Microsoft Win32 dynamic-link library that assists in device driver installation on Windows NT-based operating systems. A different co-installer file is needed for installing the checked build of a driver. The free version co-installer should not be packaged interchangeably with the checked version. A checked build of a KMDF driver cannot be installed on a free build of the OS.

These KMDF redistributable binaries are architecture dependent and are all distributed as part of the Windows Driver Kit. They can be found in the <*WDKInstallation*>\<*BuildNumber*>\redist\wdf\<*architecture*> directory.

c. Driver Catalog file (.cat file)

A .cat file contains a cryptographic hash of each file in the driver package. Windows uses these hashes to verify that the package has not been altered after it has been published. The software publisher offers authenticity by digitally signing all the files in the driver package.

The mandatory kernel-mode code-signing policy applies to all kernel-mode software and drivers for x64-based systems that are running Windows Vista. Kernel-mode signatures are also verified on 32-bit system but an administrator has the capability of installing an unsigned driver.

E. PROBLEMS ENCOUNTERES

1. Windows Driver Development

a. Windows Operating System

Despite having more than ninety percent of the market share for commercial consumer Operating Systems [27], Microsoft’s Windows has perhaps the least open literature available for understanding how the clock works beneath its desktop graphical user interface; this is Microsoft’s attempt to protect the Microsoft Windows’ proprietary. Since the MemReader driver interacts with the Physical Memory segment, a more thorough understanding of the Window’s Memory Model is needed to develop a smooth working driver.

When mapping the physical memory using the native windows API call, ZwMapViewOfSection [28], a page protection attribute needs to be provided. This specifies the type of protection for the region of initially committed pages. This attribute determines the memory type marked by Windows in a page table entry. The memory type tells the processor how to handle accesses to a page. This affects how the memory locations are read and written from the different caches (L1, L2 and L3 cache). The behavior of the Operating System becomes indeterminate when a single physical page is mapped to more than one virtual address space with conflicting memory type [45].

The anomaly described above exists in versions of the Windows Operating system before Windows XP. Versions of Windows after XP check for page mapping with different memory types. An attempt to map a page with a second memory type will result in a failure with the STATUS_CONFLICTING_ADDRESSES (0xC0000018 [29] [30]) error message.

b. Driver Development and its API

One of the design goals of the Windows Driver Foundation (WDF) [31], is to provide a consistent model to ensure that knowledge gained in the development of one driver can be transferred across device classes. This does not however reduce the copious amount of familiarization that one needs before being able to embark on driver development. With this being said, the bulk of the information available on Windows driver development can only be found at the MSDN [32] and WHDC [14] website. Each topic in driver development is written and maintained in separate papers. As such, there is largely no consistency amongst the content of the papers and not all the papers are updated with the most current material.

The community involved in driver development is also largely a small one. Sources of information are scarce especially when it comes to the specifics of dealing with the Driver Development Framework or kernel programming in Windows. The situation is not helped by the changes that are brought about with each new version of the Windows Operating system. Each of the different versions of the Windows kernel operates slightly differently and the kernel APIs are subtly different in each of the versions. Since many kernel drivers are developed to support commercial hardware products, it may not be in a company’s best interest to aide a competing company in software development.

2. Testing and Debugging

The complexity and the intertwined nature of how device drivers depend on each other to operate, with many device drivers working together and depending of each other, greatly complicate the job of testing and debugging a device driver. It is difficult to determine the source of error and explain anomalies and erroneous behavior during testing.

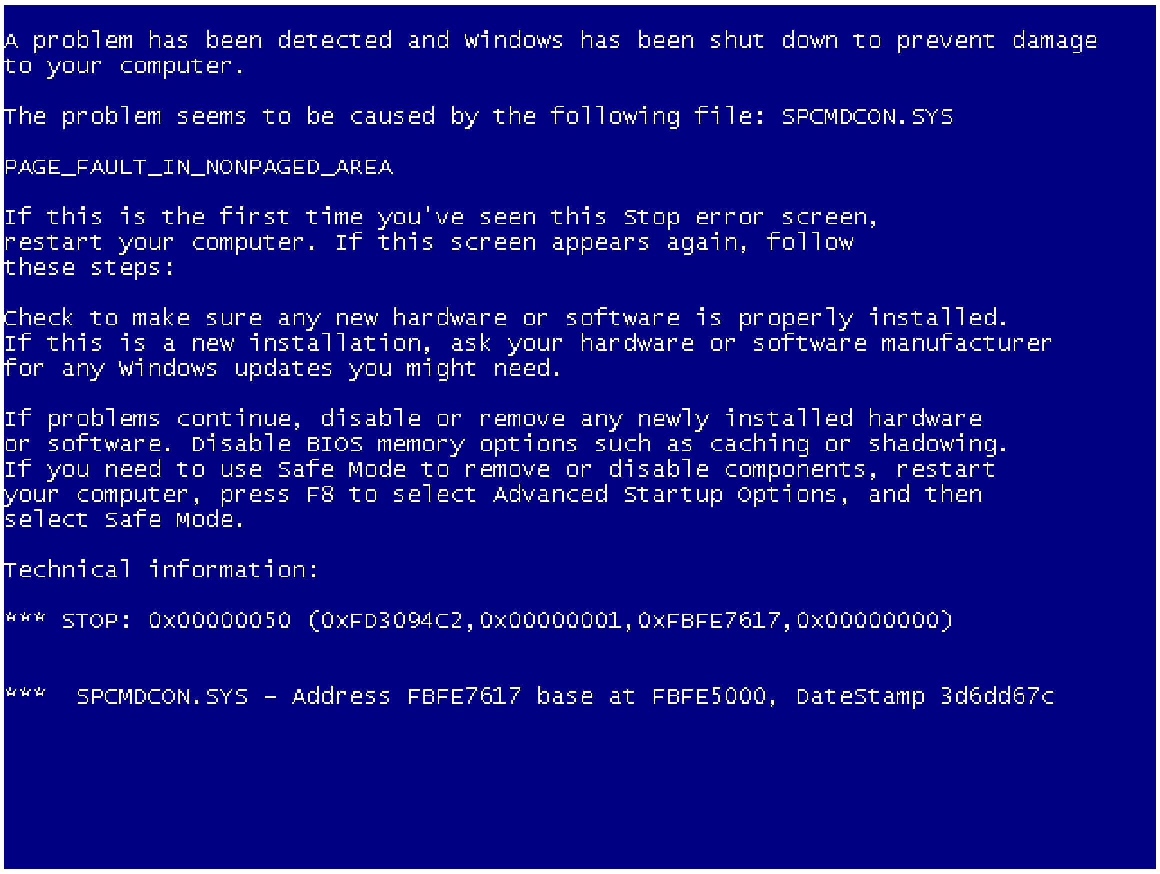
a. Blue Screen of Death (BSoD)

It is often difficult to recover from errors and faults in an Operating System’s kernel components. Faults or errors usually result in a system crash or in Microsoft Windows Operating System’s case, a Blue Screen of Death (BSoD). This is also sometimes referred as a system halt or bug check. When this happens, Windows halts its operations and creates a memory dump file with information on the state of the system when the error occurred. There is also some information displayed on the Blue Screen that describes the error [33].

The first section shows the actual error message. The error messages may give a clue regarding the source of the error, but messages like BAD_POOL_HEADER are cryptic and they do not really help with pin pointing the source of error. Specific technical information regarding the crash can be found at the bottom of the screen in a section aptly titled ‘Technical Information’.

Figure 2 is a screen capture of a BSoD for a Page Fault In Non Paged Area bug check. The hexadecimal code that follows the word ‘STOP’ in the Technical Information section is the bug check code used by the Operating

System. The four parameters that follow contain specific information regarding this instance of bug check. The name of the component, if available, that caused this error is also listed. The SPCMDCON.SYS is the device driver that caused the bug check in Figure 2. The bug check code can be used as a reference key [34] to find specific information on each parameter.



```
A problem has been detected and windows has been shut down to prevent damage
to your computer.

The problem seems to be caused by the following file: SPCMDCON.SYS

PAGE_FAULT_IN_NONPAGED_AREA

If this is the first time you've seen this Stop error screen,
restart your computer. If this screen appears again, follow
these steps:

Check to make sure any new hardware or software is properly installed.
If this is a new installation, ask your hardware or software manufacturer
for any Windows updates you might need.

If problems continue, disable or remove any newly installed hardware
or software. Disable BIOS memory options such as caching or shadowing.
If you need to use Safe Mode to remove or disable components, restart
your computer, press F8 to select Advanced Startup Options, and then
select Safe Mode.

Technical information:

*** STOP: 0x00000050 (0xFD3094C2,0x00000001,0xFBFE7617,0x00000000)


***  SPCMDCON.SYS - Address FBFE7617 base at FBFE5000, DateStamp 3d6dd67c
```

Figure 2. Blue Screen of Death: Page_Fault_In_Non_Paged_Area

b. ZwMapViewOfSection [28]

The ZwMapViewOfSection routine maps a view of a section of the Physical Memory into the virtual address space of a subject process. Ten parameters need to be provided to create a successful mapping. ZwMapViewOfSection returns a NTSTATUS value to report on the status of the mapping operation. ZwMapViewOfSection is a nontrivial mapping function and an unsuccessful mapping can be due to many reasons. It can be exasperating when trying to determine the cause of a failed operation.

There are many return values for NTSTATUS that will help determine the cause of an unsuccessful mapping. It is important to capture the specific returned status in the event of an unsuccessful mapping. The description of each specific error code can be found in the ntstatus.h file in the win32api package and it is readily available on the Internet [30]. It is useful to include the error code and/ or the error description when searching, for example on Google.com, to find the meaning of an error return value and a possible approach to mitigate the error.

In the earlier stages of testing, there was a consistent discrepancy in the location of the first appearance of the known string. These known strings are deliberately injected into RAM as markers. They will be discussed in more detail in the chapter on test methodology. There was always a misalignment of 8 times the TX_BLOCK_SIZE constant (see public.h in Appendix E). Upon further examination, this misalignment was attributed to the failed mappings of the ZwMapViewOfSection. In the event of a read failure from the MemReader driver, the MemReader application will skip this write operation for the cycle and proceed on to the next offset. Instead of skipping the write operation, the MemReader will need to pad the image file with NULL bytes[7] so that the two images (the control image created by ManTech's DD or MDD and the image created by MemReader) maintained offset synchronization. This also leads to the use of the same TX_BLOCK_SIZE as MDD.

[7] This operation is similar to the conv=sync option in the linux tool dd.

THIS PAGE INTENTIONALLY LEFT BLANK

V. TEST METHODOLOGY

A. TESTING OVERVIEW

The content of RAM is changing every minute, with or without any human intervention, as many background processes are executing to support the Operating System. When the any tool or application is executed, it too will be loaded into memory thereby changing the contents of RAM.

The testing to show that both the developed driver and application is extracting a correct image of RAM can be split into two parts as described the following section.

B. TESTING FOR THE PRESENCE OF DATA

A certain amount of known information was injected into memory and the image created by the developed driver (and tool) was inspected for the presence of that information .The simplest form of information to inject and detect would be ASCII encoded strings.

This can be achieved by opening the notepad application and typing many instances of a string of characters on a Windows system that is loaded with the MemReader driver. The binary representation of the repeated string must be stored in some part of memory. Without exiting the notepad application, start the MemReader tool to create an image of the system's RAM. If the MemReader has correctly imaged the RAM, the repeated string of characters should also be found in the created image file.

The image file can then be inspected with a HEX/ ASCII editor [11] [12] that is capable of displaying both the binary form of the data and any ASCII encoded information. If the extraction and creation of the image file is done correctly, the repeated string of characters should be correctly displayed in the editor.

C. TESTING BY COMPARISON

The testing approach described in Section B in this chapter is a simple, functional test to prove that the driver and imaging tool is working. However, the test only shows that MemReader has correctly imaged the part of memory used by an active process (notepad). In comparison with the size of memory in systems today, application memory only accounts for a small percentage of the memory, and does not provide for any method of verifying addresses, page alignment, etc.

A more definitive method of testing is by comparing two memory images, one that is created by a known good means and the other by MemReader. Since the content of RAM is consistently changing due to a “sliding window” effect [3], it will be impossible to have two images that have a 100% match. A high degree of congruency between the images is however expected.

1. RAM Image Obtained Using a Known Good Tool

For the purpose of this research, the comparison was made with the image created by the ManTech DD (MDD) tool. The ManTech DD was an obvious candidate for the following reason:

a. The source code was accessible and understood so that there was no ambiguity about the format of the data image created by MDD.

b. To date, not many other software tools are available for creating an image of the RAM for the Windows Operating System.

To ensure minimal difference between the two images, created using MemReader and the ManTech DD, the system should be very lightly loaded during the test and the two tools should be run, one immediately followed by the other. A more detailed discussion on the expected content of memory is done in the next chapter.

2. RAM Image Obtained Using VMware

The use of VMWare [10] creates a pristine image of the Operating System when the Virtual Machine (VM) is paused. If the VM is loaded with the MemReader driver, an image of the RAM can be captured immediately after the VM is restarted. This ensures minimal changes between the two images. Since this method does not require any instrumentation in the VM itself (the action is entirely external), the delta between the VMWare memory file and the acquired file are expected to be smaller than the delta present in method 1.

D. TESTING ENVIRONMENT AND SETUP

The test target is a Dell Optiplex GX280 Series system running on the Windows Vista Business Operating System. The machine specification is listed in Table 3. The RAM capacity was obtain programmatically using Window native API GlobalMemoryStatusEx [42].

	Description
Processor	Intel® Pentium® 4 CPU 3.40GHz 3.39GHz
Memory (RAM)	2037.52MB
System Type	32-bit Operating System
Operating System	Windows Vista Business

Table 3. Test Target System Specification

In an ideal situation, images should be created by the several testing methods so that the information of the physical memory is the same for each method. A set of procedure are followed in order to ensure that the data in the physical memory is as similar as possible on each test of the system when the different RAM images are created. Upon system start up, the same user-account is used. The system is left to idle for a five-minute period for the system applications and service to be loaded, this includes the MemReader service driver. A text file containing repeated known strings as described in subsection B is opened using notepad. The RAM image file is first created using the

ManTech’s DD tool. Subsequently, another RAM image is created using the MemReader application. For more detailed information on the procedures for creating a RAM image see Appendix D.

VI. RESULTS AND ANALYSIS

A. DEVELOPMENT RESULTS

The MemReader device driver developed is still an unfinished product. There are still a few bugs and there are still some issues with its stability that can be further explored in future works. MemReader requires some fine-tuning and arguably better integration with the Windows operating system in order to achieve enough stability for production use. The testing methodology proposed in the previous chapter, therefore, cannot currently be strictly adhered to.

The development still yielded an installable MemReader driver package and a user-mode MemReader application that utilizes and interfaces with the MemReader driver and extracts information from the RAM. The MemReader driver installs under the Class Name 'Tools' with Service Description name "Memory Reader Service', specified in the memreader.inx file (see Appendix E).

The MemReader application takes in two parameters, namely the file name of the image in which the extracted information will be saved and an optional offset (in Megabytes) where the driver will start reading memory information from. The following example would create an image file name test.bin from starting one megabyte into memory:

```
>MemReader.exe test.bin 1
```

The first observation from running the MemReader application on a system installed with the MemReader Service driver was that the application was unable to extract information at the sixth megabytes offset. This happened in all the test runs and the MemReader application fails with a system error 487. A simple lookup [38] explains that the application made an "attempt to access (an) invalid address". Debugging the MemReader service driver revealed that the routine ZwMapViewOfSection has failed with a NTSTATUS error code 0xC0000018 – STATUS_CONFLICTING_ADDRESSES. The MemReader application ignores this error and continues extracting information from other memory locations by skipping this offset. The MemReader driver is however still somewhat unstable and can cause indeterminate system crashes during the process of extracting data from RAM. The cause of these crashes could not be determined.

B. MODIFIED TESTING METHODOLOGY

A RAM image extracted using the ManTech's DD (MDD) tool was created and used as a control for which the other extracted image, using the MemReader application and MemReader service driver, were verified against. There is still a stability issue with the MemReader service driver and it causes indeterminate system halts during the creation of the test RAM image. Although a complete image of RAM is not captured, the lower regions (offset) are captured[8] and can be used for verification purpose.

Two text files containing the repetition of two different known strings are opened during testing. The two text files, that are opened using two separate instances of the notepad application, will occupy two separate region of memory. This was a useful marker for the verification of the created test image. The details of the two text files are as specified in Table 4.

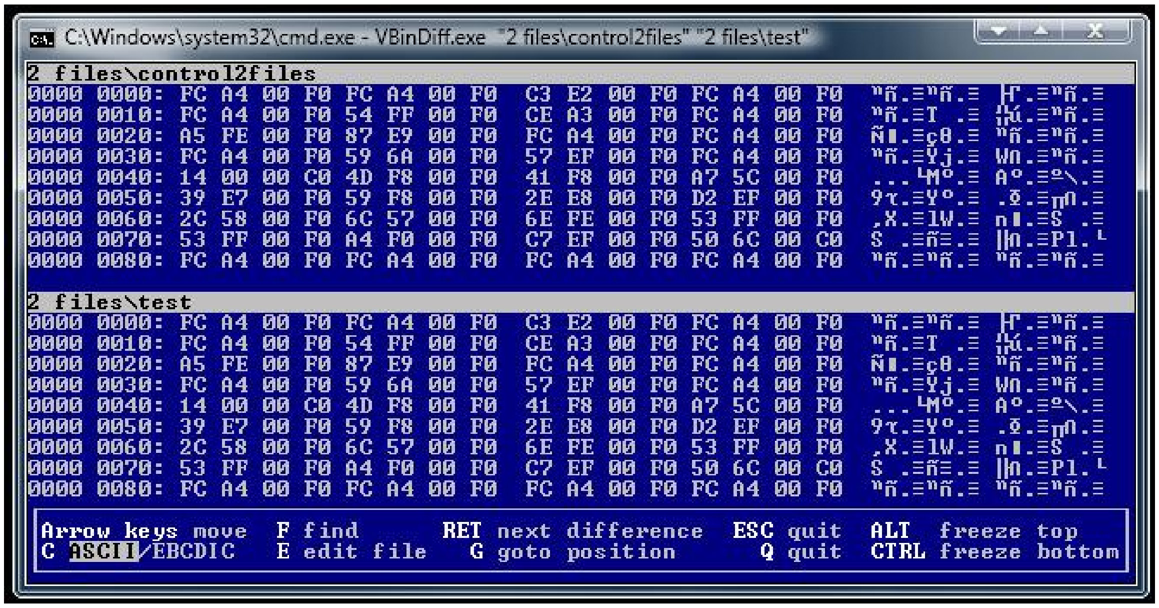
File Name	TestFile1.txt	TestFile2.txt
File Size	16,391KB	16,393KB
Repeated String	This is an usually short string for such a big file.	the quick brown fox jumped over the lazy dog.
String count	316426 (1 incomplete)	364525 (1 incomplete)

Table 4. Text files of repeated known strings

An open source application, Visual Binary Diff [12], was used to compare the two images created. Visual Binary Diff or VBinDiff opens the two files simultaneously and dumps their hex and ASCII outputs one on top of the other.

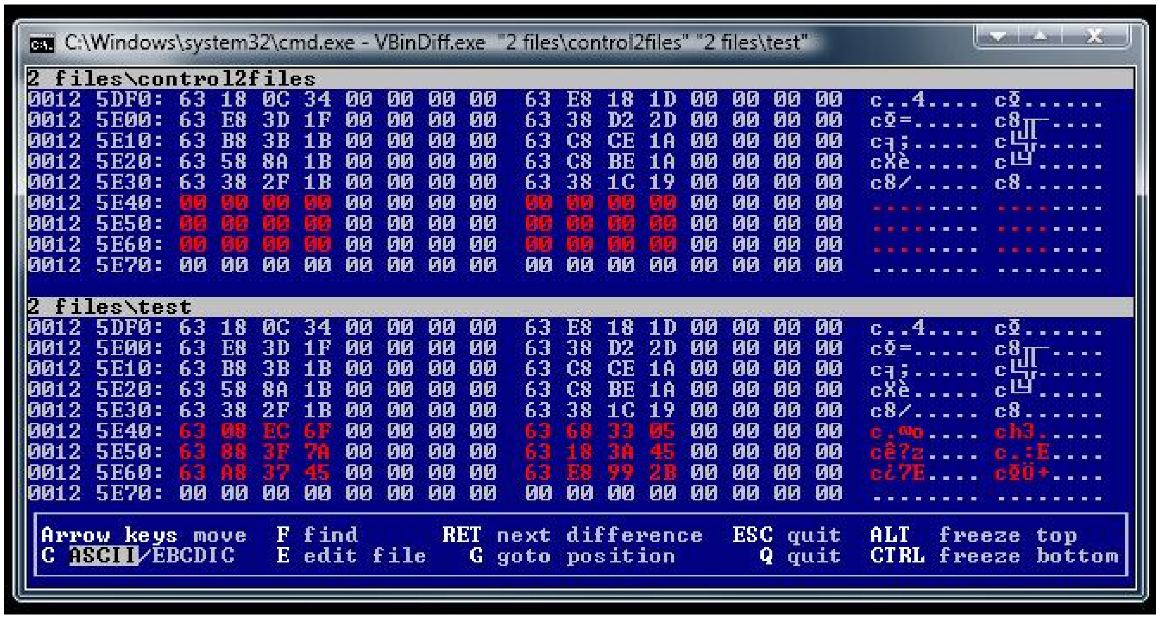
[8] Captures of greater than 1,600 MB's have been achieved.

Differences between the two file are marked red. Figure 3 and 4 are screenshots of the VBinDiff showing how the information is presented.



```
C:\Windows\system32\cmd.exe - VBinDiff.exe "2 files\control2files" "2 files\test"
2 files\control2files
0000 0000: FC A4 00 F0 FC A4 00 F0  C3 E2 00 F0 FC A4 00 F0  ⁿñ.≡ⁿñ.≡ ├Γ.≡ⁿñ.≡
0000 0010: FC A4 00 F0 54 FF 00 F0  CE A3 00 F0 FC A4 00 F0  ⁿñ.≡T .≡ ╬ú.≡ⁿñ.≡
0000 0020: A5 FE 00 F0 87 E9 00 F0  FC A4 00 F0 FC A4 00 F0  Ñ■.≡çΘ.≡ ⁿñ.≡ⁿñ.≡
0000 0030: FC A4 00 F0 59 6A 00 F0  57 EF 00 F0 FC A4 00 F0  ⁿñ.≡Yj.≡ W∩.≡ⁿñ.≡
0000 0040: 14 00 00 C0 4D F8 00 F0  41 F8 00 F0 A7 5C 00 F0  ...└M°.≡ A°.≡º\.≡
0000 0050: 39 E7 00 F0 59 F8 00 F0  2E E8 00 F0 D2 EF 00 F0  9τ.≡Y°.≡ .Φ.≡╥∩.≡
0000 0060: 2C 58 00 F0 6C 57 00 F0  6E FE 00 F0 53 FF 00 F0  ,X.≡lW.≡ n■.≡S .≡
0000 0070: 53 FF 00 F0 A4 F0 00 F0  C7 EF 00 F0 50 6C 00 C0  S .≡ñ≡.≡ ╟∩.≡Pl.└
0000 0080: FC A4 00 F0 FC A4 00 F0  FC A4 00 F0 FC A4 00 F0  ⁿñ.≡ⁿñ.≡ ⁿñ.≡ⁿñ.≡

2 files\test
0000 0000: FC A4 00 F0 FC A4 00 F0  C3 E2 00 F0 FC A4 00 F0  ⁿñ.≡ⁿñ.≡ ├Γ.≡ⁿñ.≡
0000 0010: FC A4 00 F0 54 FF 00 F0  CE A3 00 F0 FC A4 00 F0  ⁿñ.≡T .≡ ╬ú.≡ⁿñ.≡
0000 0020: A5 FE 00 F0 87 E9 00 F0  FC A4 00 F0 FC A4 00 F0  Ñ■.≡çΘ.≡ ⁿñ.≡ⁿñ.≡
0000 0030: FC A4 00 F0 59 6A 00 F0  57 EF 00 F0 FC A4 00 F0  ⁿñ.≡Yj.≡ W∩.≡ⁿñ.≡
0000 0040: 14 00 00 C0 4D F8 00 F0  41 F8 00 F0 A7 5C 00 F0  ...└M°.≡ A°.≡º\.≡
0000 0050: 39 E7 00 F0 59 F8 00 F0  2E E8 00 F0 D2 EF 00 F0  9τ.≡Y°.≡ .Φ.≡╥∩.≡
0000 0060: 2C 58 00 F0 6C 57 00 F0  6E FE 00 F0 53 FF 00 F0  ,X.≡lW.≡ n■.≡S .≡
0000 0070: 53 FF 00 F0 A4 F0 00 F0  C7 EF 00 F0 50 6C 00 C0  S .≡ñ≡.≡ ╟∩.≡Pl.└
0000 0080: FC A4 00 F0 FC A4 00 F0  FC A4 00 F0 FC A4 00 F0  ⁿñ.≡ⁿñ.≡ ⁿñ.≡ⁿñ.≡

 Arrow keys move  F find     RET next difference  ESC quit  ALT  freeze top
 C ASCII/EBCDIC   E edit file  G goto position      Q quit  CTRL freeze bottom
```

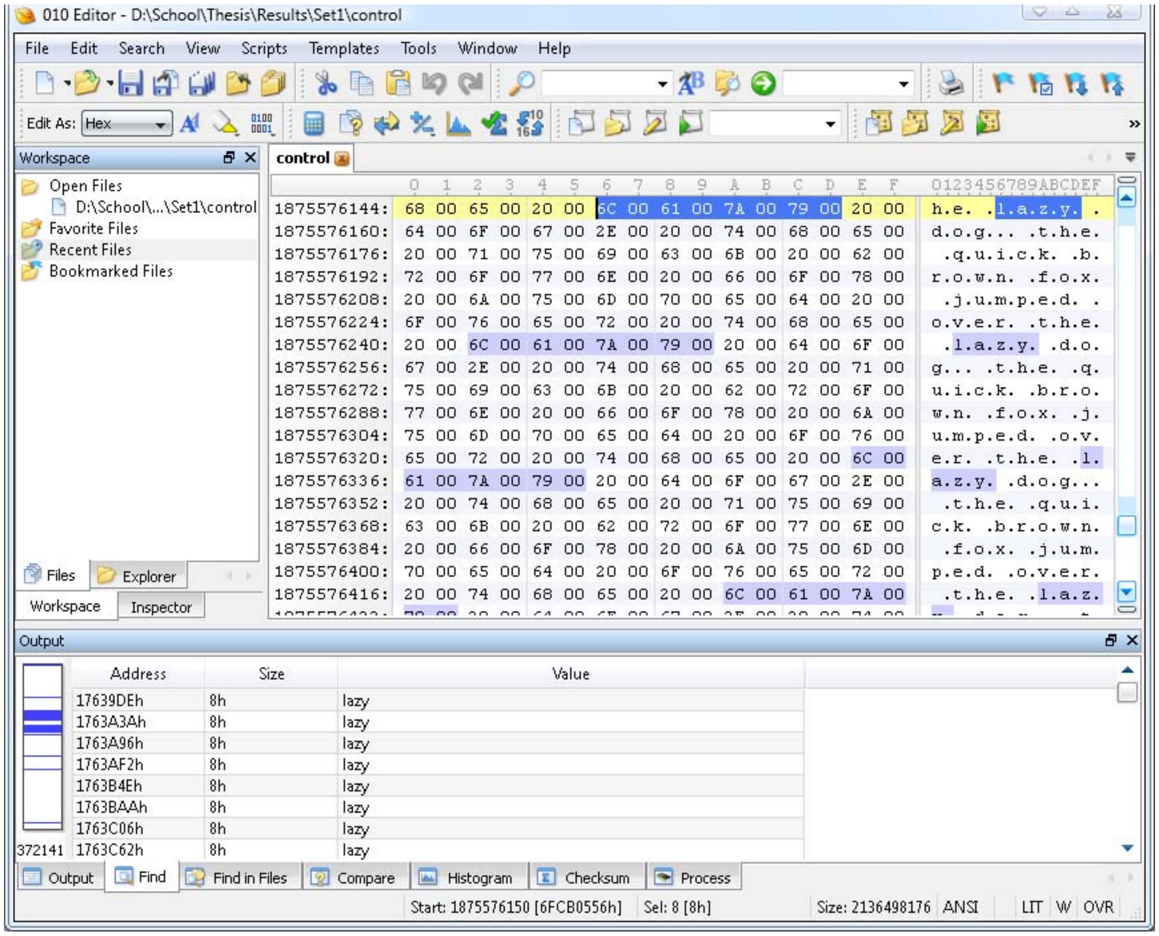
Figure 3. VBinDiff output – Not difference in data



```
C:\Windows\system32\cmd.exe - VBinDiff.exe "2 files\control2files" "2 files\test"
2 files\control2files
0012 5DF0: 63 18 0C 34 00 00 00 00  63 E8 18 1D 00 00 00 00  c..4.... cΦ......
0012 5E00: 63 E8 3D 1F 00 00 00 00  63 38 D2 2D 00 00 00 00  cΦ=..... c8╥-....
0012 5E10: 63 B8 3B 1B 00 00 00 00  63 C8 CE 1A 00 00 00 00  c╕;..... c╚╬.....
0012 5E20: 63 58 8A 1B 00 00 00 00  63 C8 BE 1A 00 00 00 00  cXè..... c╚╛.....
0012 5E30: 63 38 2F 1B 00 00 00 00  63 38 1C 19 00 00 00 00  c8/..... c8......
0012 5E40: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0012 5E50: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0012 5E60: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0012 5E70: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........

2 files\test
0012 5DF0: 63 18 0C 34 00 00 00 00  63 E8 18 1D 00 00 00 00  c..4.... cΦ......
0012 5E00: 63 E8 3D 1F 00 00 00 00  63 38 D2 2D 00 00 00 00  cΦ=..... c8╥-....
0012 5E10: 63 B8 3B 1B 00 00 00 00  63 C8 CE 1A 00 00 00 00  c╕;..... c╚╬.....
0012 5E20: 63 58 8A 1B 00 00 00 00  63 C8 BE 1A 00 00 00 00  cXè..... c╚╛.....
0012 5E30: 63 38 2F 1B 00 00 00 00  63 38 1C 19 00 00 00 00  c8/..... c8......
0012 5E40: 63 08 EC 6F 00 00 00 00  63 68 33 05 00 00 00 00  c.∞o.... ch3.....
0012 5E50: 63 88 3F 7A 00 00 00 00  63 18 3A 45 00 00 00 00  cê?z.... c.:E....
0012 5E60: 63 A8 37 45 00 00 00 00  63 E8 99 2B 00 00 00 00  c¿7E.... cΦÖ+....
0012 5E70: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........

 Arrow keys move  F find     RET next difference  ESC quit  ALT  freeze top
 C ASCII/EBCDIC   E edit file  G goto position      Q quit  CTRL freeze bottom
```

Figure 4. VBinDiff – Difference in data marked in red

A trial version of a Text/ Hex editor, 010 Editor [11] was used to inspect the image files to locate traces of the known strings. The editor is capable of display the hex and ASCII representation beside each other, and has powerful features such as searching for ASCII, EBCDIC, Unicode strings. Figure 5 is a screenshot of the 010 Editor and it shows how the information is represented.


010 Editor - D:\School\Thesis\Results\Set1\control
File Edit Search View Scripts Templates Tools Window Help
Edit As: Hex
Workspace
Open Files
D:\School\...\Set1\control
Favorite Files
Recent Files
Bookmarked Files
Files Explorer
Workspace Inspector
control
Output
Output Find Find in Files Compare Histogram Checksum Process

```
             0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
1875576144: 68 00 65 00 20 00 6C 00 61 00 7A 00 79 00 20 00   h.e. .l.a.z.y. .
1875576160: 64 00 6F 00 67 00 2E 00 20 00 74 00 68 00 65 00   d.o.g... .t.h.e.
1875576176: 20 00 71 00 75 00 69 00 63 00 6B 00 20 00 62 00    .q.u.i.c.k. .b.
1875576192: 72 00 6F 00 77 00 6E 00 20 00 66 00 6F 00 78 00   r.o.w.n. .f.o.x.
1875576208: 20 00 6A 00 75 00 6D 00 70 00 65 00 64 00 20 00    .j.u.m.p.e.d. .
1875576224: 6F 00 76 00 65 00 72 00 20 00 74 00 68 00 65 00   o.v.e.r. .t.h.e.
1875576240: 20 00 6C 00 61 00 7A 00 79 00 20 00 64 00 6F 00    .l.a.z.y. .d.o.
1875576256: 67 00 2E 00 20 00 74 00 68 00 65 00 20 00 71 00   g... .t.h.e. .q.
1875576272: 75 00 69 00 63 00 6B 00 20 00 62 00 72 00 6F 00   u.i.c.k. .b.r.o.
1875576288: 77 00 6E 00 20 00 66 00 6F 00 78 00 20 00 6A 00   w.n. .f.o.x. .j.
1875576304: 75 00 6D 00 70 00 65 00 64 00 20 00 6F 00 76 00   u.m.p.e.d. .o.v.
1875576320: 65 00 72 00 20 00 74 00 68 00 65 00 20 00 6C 00   e.r. .t.h.e. .l.
1875576336: 61 00 7A 00 79 00 20 00 64 00 6F 00 67 00 2E 00   a.z.y. .d.o.g...
1875576352: 20 00 74 00 68 00 65 00 20 00 71 00 75 00 69 00    .t.h.e. .q.u.i.
1875576368: 63 00 6B 00 20 00 62 00 72 00 6F 00 77 00 6E 00   c.k. .b.r.o.w.n.
1875576384: 20 00 66 00 6F 00 78 00 20 00 6A 00 75 00 6D 00    .f.o.x. .j.u.m.
1875576400: 70 00 65 00 64 00 20 00 6F 00 76 00 65 00 72 00   p.e.d. .o.v.e.r.
1875576416: 20 00 74 00 68 00 65 00 20 00 6C 00 61 00 7A 00    .t.h.e. .l.a.z.
```

```
        Address    Size  Value
        17639DEh   8h    lazy
        1763A3Ah   8h    lazy
        1763A96h   8h    lazy
        1763AF2h   8h    lazy
        1763B4Eh   8h    lazy
        1763BAAh   8h    lazy
        1763C06h   8h    lazy
372141  1763C62h   8h    lazy
```

```
Start: 1875576150 [6FCB0556h]   Sel: 8 [8h]   Size: 2136498176   ANSI   LIT   W   OVR
```

Figure 5. VBinDiff – Difference in data marked in red

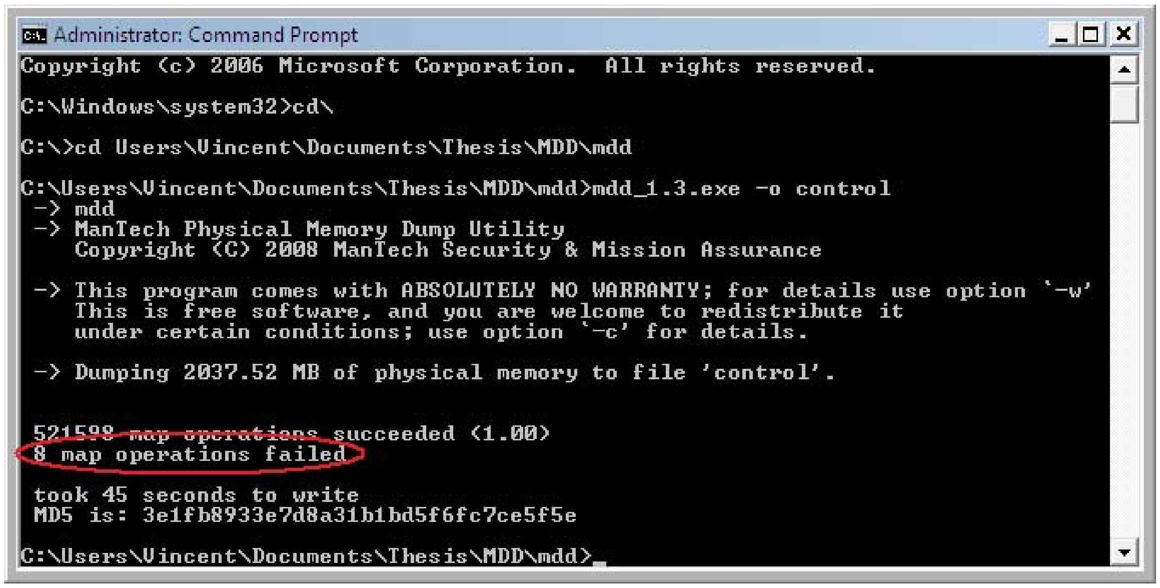
C. TESTING RESULTS

Three sets of images were collected and analyzed. A summary of the images collected is found in Table 5. The transfer block size defined by the TX_BLOCK_SIZE constant in the file public.h (See Appendix E) is the equivalent of the block size parameter of the GNU dd tool [43]. The MemReader driver and application needs to be rebuilt if the TX_BLOCK_SIZE is changed.

	Set 1	Set 2	Set 3
Control Image File Size	2,086,424KB (~2037.5MB)	2,086,424KB	2,086,424KB
Test Image File Size	1,638,464KB	1,263,936KB	587,904KB
Transfer Block Size	4096bytes	4096bytes	4096bytes

Table 5. Test Results

The control image file size created was the same for the all three collections. MDD, however, did not manage to extract information from certain regions of memory. The MDD output shows that there are 8 failed mappings during the creation of the control image. See Figure 6. These failed mapping would not have resulted in a smaller image as MDD will pad the images at locations where the mapping failed. This is important as the MemReader application should behave in the same way so that it can be compared against the images created by MDD.



```
Copyright (c) 2006 Microsoft Corporation.  All rights reserved.

C:\Windows\system32>cd\

C:\>cd Users\Vincent\Documents\Thesis\MDD\mdd

C:\Users\Vincent\Documents\Thesis\MDD\mdd>mdd_1.3.exe -o control
 -> mdd
 -> ManTech Physical Memory Dump Utility
    Copyright (C) 2008 ManTech Security & Mission Assurance

 -> This program comes with ABSOLUTELY NO WARRANTY; for details use option `-w'
    This is free software, and you are welcome to redistribute it
    under certain conditions; use option `-c' for details.

 -> Dumping 2037.52 MB of physical memory to file 'control'.


 521598 map operations succeeded (1.00)
 8 map operations failed

 took 45 seconds to write
 MD5 is: 3e1fb8933e7d8a31b1bd5f6fc7ce5f5e

C:\Users\Vincent\Documents\Thesis\MDD\mdd>_
```

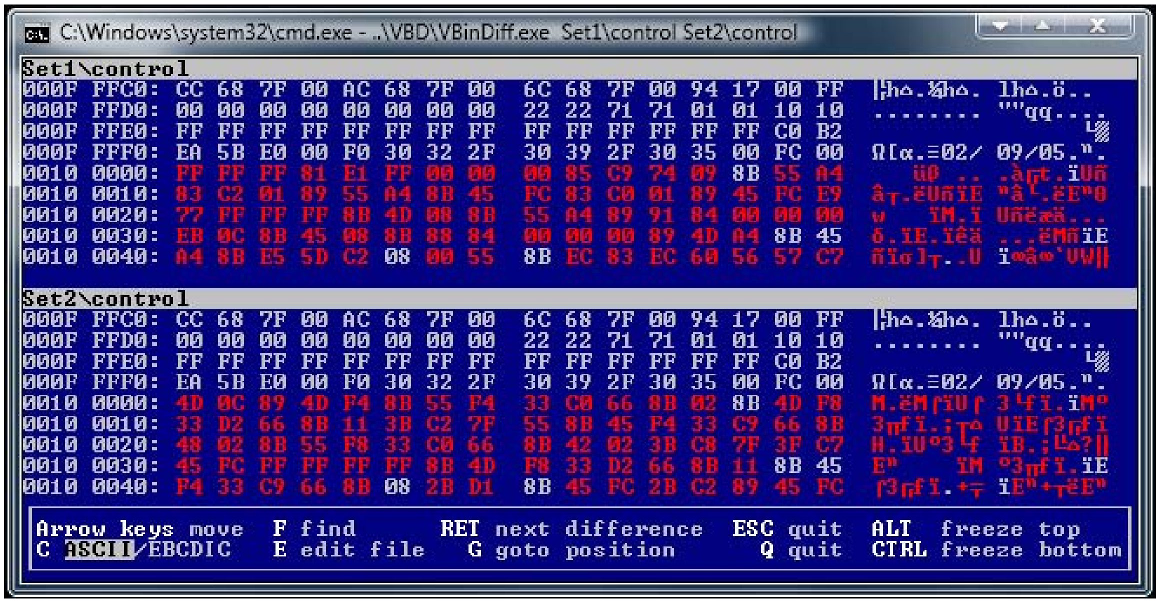
Figure 6. MDD output screen when creating control image

D. ANALYSIS OF RESULTS

1. Preliminary Analysis

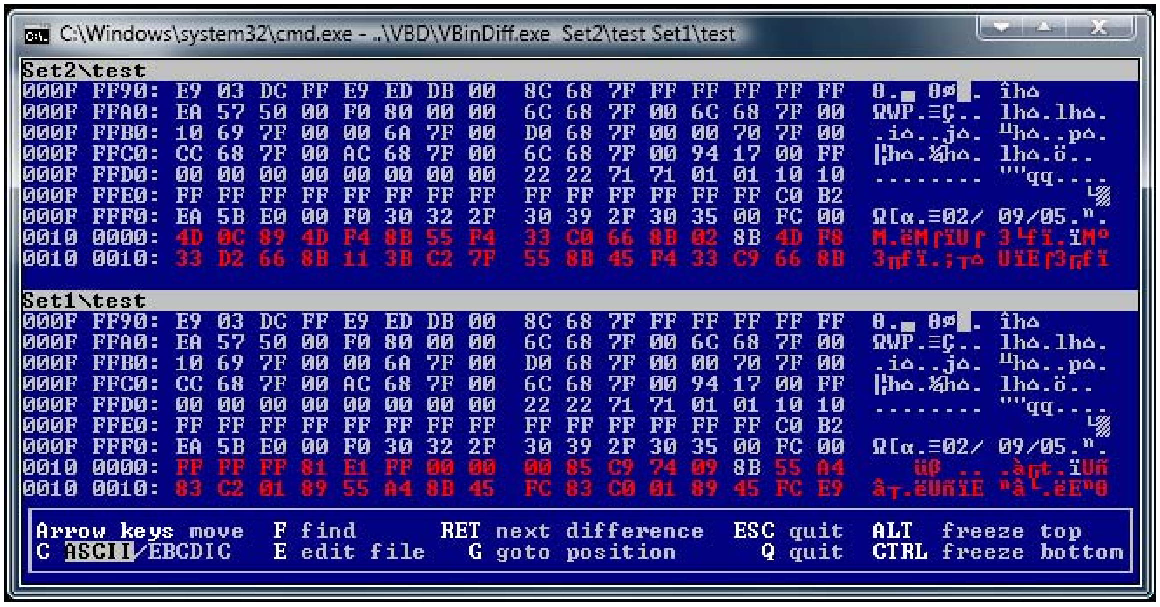
The first significant observation was that even with the MDD, a complete image of RAM could not be extracted. The blocks that MDD was unable to extract could possibly be the regions which the MemReader driver gets a STATUS_CONFLICTING_ADDRESSES error.

A comparison was done between the control images collected. Upon visual inspection using VBinDiff, the memory segment from 0x00000000 to 0x000FFFF, the first megabyte, remained largely the same. This is expected as most of the system information would have been the first to be loaded into the RAM and this information should remain largely the same. This was consistent with the comparison between the test images. Significant differences began only at the one megabyte offset. Figure 7 shows the VBinDiff differences between two control images. Figure 8 shows the VBinDiff differences between two test images.



```
C:\Windows\system32\cmd.exe - ..\VBD\VBinDiff.exe  Set1\control Set2\control
Set1\control
000F FFC0: CC 68 7F 00 AC 68 7F 00  6C 68 7F 00 94 17 00 FF  ╠h⌂.¼h⌂. lh⌂.ö..
000F FFD0: 00 00 00 00 00 00 00 00  22 22 71 71 01 01 10 10  ........ ""qq....
000F FFE0: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF C0 B2                 └▓
000F FFF0: EA 5B E0 00 F0 30 32 2F  30 39 2F 30 35 00 FC 00  Ω[α.≡02/ 09/05.ⁿ.
0010 0000: FF FF FF 81 E1 FF 00 00  00 85 C9 74 09 8B 55 A4     üß ..  .à╔t.ïUñ
0010 0010: 83 C2 01 89 55 A4 8B 45  FC 83 C0 01 89 45 FC E9  â┬.ëUñïE ⁿâ└.ëEⁿΘ
0010 0020: 77 FF FF FF 8B 4D 08 8B  55 A4 89 91 84 00 00 00  w   ïM.ï Uñëæä...
0010 0030: EB 0C 8B 45 08 8B 88 84  00 00 00 89 4D A4 8B 45  δ.ïE.ïêä ...ëMñïE
0010 0040: A4 8B E5 5D C2 08 00 55  8B EC 83 EC 60 56 57 C7  ñïσ]┬..U ï∞â∞`VW╟

Set2\control
000F FFC0: CC 68 7F 00 AC 68 7F 00  6C 68 7F 00 94 17 00 FF  ╠h⌂.¼h⌂. lh⌂.ö..
000F FFD0: 00 00 00 00 00 00 00 00  22 22 71 71 01 01 10 10  ........ ""qq....
000F FFE0: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF C0 B2                 └▓
000F FFF0: EA 5B E0 00 F0 30 32 2F  30 39 2F 30 35 00 FC 00  Ω[α.≡02/ 09/05.ⁿ.
0010 0000: 4D 0C 89 4D F4 8B 55 F4  33 C0 66 8B 02 8B 4D F8  M.ëM⌠ïU⌠ 3└fï.ïM°
0010 0010: 33 D2 66 8B 11 3B C2 7F  55 8B 45 F4 33 C9 66 8B  3╥fï.;┬⌂ UïE⌠3╔fï
0010 0020: 48 02 8B 55 F8 33 C0 66  8B 42 02 3B C8 7F 3F C7  H.ïU°3└f ïB.;╚⌂?╟
0010 0030: 45 FC FF FF FF FF 8B 4D  F8 33 D2 66 8B 11 8B 45  Eⁿ    ïM °3╥fï.ïE
0010 0040: F4 33 C9 66 8B 08 2B D1  8B 45 FC 2B C2 89 45 FC  ⌠3╔fï.+╤ ïEⁿ+┬ëEⁿ

 Arrow keys move   F find       RET next difference   ESC quit   ALT  freeze top
 C ASCII/EBCDIC    E edit file    G goto position       Q quit   CTRL freeze bottom
```

Figure 7. VBinDiff – Difference in data between Set1\control and Set2\control

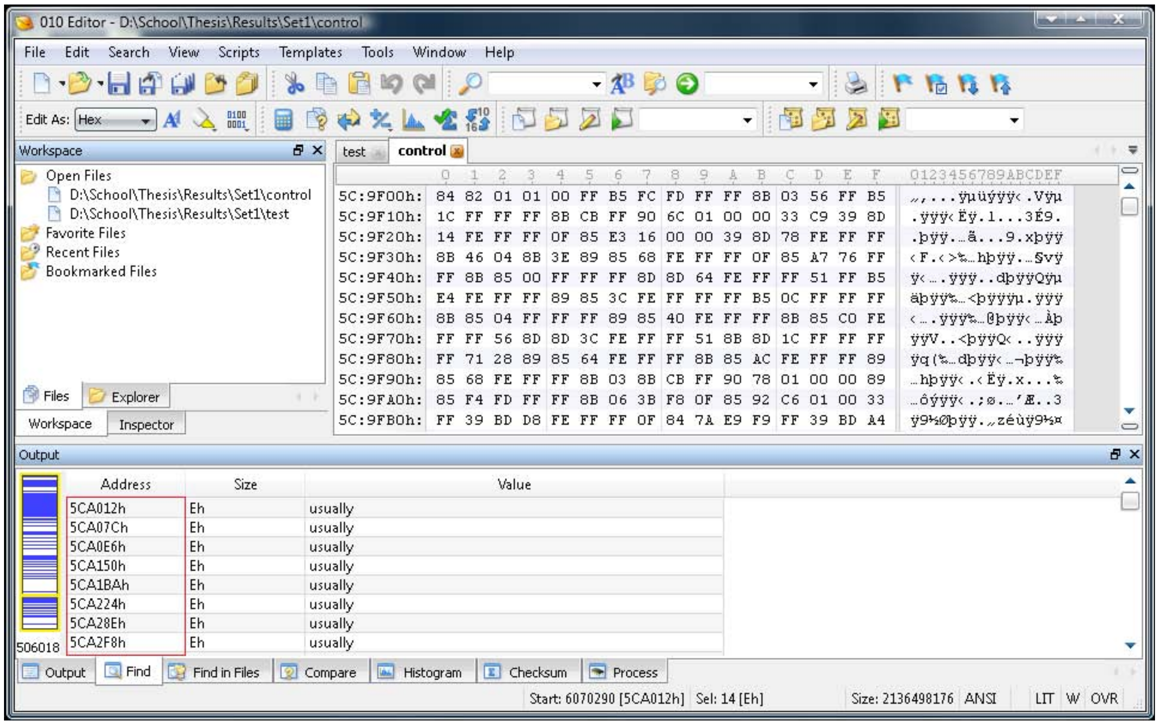


```
C:\Windows\system32\cmd.exe - ..\VBD\VBinDiff.exe  Set2\test Set1\test
Set2\test
000F FF90: E9 03 DC FF E9 ED DB 00  8C 68 7F FF FF FF FF FF  θ.▄ θø█. îh⌂
000F FFA0: EA 57 50 00 F0 80 00 00  6C 68 7F 00 6C 68 7F 00  ΩWP.≡Ç.. lh⌂.lh⌂.
000F FFB0: 10 69 7F 00 00 6A 7F 00  D0 68 7F 00 00 70 7F 00  .i⌂..j⌂. ╨h⌂..p⌂.
000F FFC0: CC 68 7F 00 AC 68 7F 00  6C 68 7F 00 94 17 00 FF  ╠h⌂.¼h⌂. lh⌂.ö..
000F FFD0: 00 00 00 00 00 00 00 00  22 22 71 71 01 01 10 10  ........ ""qq....
000F FFE0: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF C0 B2                └▓
000F FFF0: EA 5B E0 00 F0 30 32 2F  30 39 2F 30 35 00 FC 00  Ω[α.≡02/ 09/05.ⁿ.
0010 0000: 4D 0C 89 4D F4 8B 55 F4  33 C0 66 8B 02 8B 4D F8  M.ëM⌠ïU⌠ 3└fï.ïM°
0010 0010: 33 D2 66 8B 11 3B C2 7F  55 8B 45 F4 33 C9 66 8B  3╥fï.;┬⌂ UïE⌠3╔fï

Set1\test
000F FF90: E9 03 DC FF E9 ED DB 00  8C 68 7F FF FF FF FF FF  θ.▄ θø█. îh⌂
000F FFA0: EA 57 50 00 F0 80 00 00  6C 68 7F 00 6C 68 7F 00  ΩWP.≡Ç.. lh⌂.lh⌂.
000F FFB0: 10 69 7F 00 00 6A 7F 00  D0 68 7F 00 00 70 7F 00  .i⌂..j⌂. ╨h⌂..p⌂.
000F FFC0: CC 68 7F 00 AC 68 7F 00  6C 68 7F 00 94 17 00 FF  ╠h⌂.¼h⌂. lh⌂.ö..
000F FFD0: 00 00 00 00 00 00 00 00  22 22 71 71 01 01 10 10  ........ ""qq....
000F FFE0: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF C0 B2                └▓
000F FFF0: EA 5B E0 00 F0 30 32 2F  30 39 2F 30 35 00 FC 00  Ω[α.≡02/ 09/05.ⁿ.
0010 0000: FF FF FF 81 E1 FF 00 00  00 85 C9 74 09 8B 55 A4     üß .. .à╔t.ïUñ
0010 0010: 83 C2 01 89 55 A4 8B 45  FC 83 C0 01 89 45 FC E9  â┬.ëUñïE ⁿâ└.ëEⁿθ

 Arrow keys move   F find        RET next difference   ESC quit   ALT  freeze top
 C ASCII/EBCDIC    E edit file     G goto position       Q quit   CTRL freeze bottom
```

Figure 8. VBinDiff – Difference in data between Set1\test and Set2\test

2. Testing for the Presence of Data

The image files were searched using parts of the known string as a "signature" and it was expected that the repeated known strings occupied multiple fragments of contiguous chunk of memory. The "signature" used to detect the TestFile1.txt and TestFile2.txt strings are "*usually*" and "*quick brown*" respectively. Figure 10 to 13 shows the result of a "find" on the two signatures in the control and test images. The box highlighted in yellow is a graphical representation of the image file. Regions marked in blue are the locations where the signatures are found. In an ideal situation, the content of the TestFiles should occupy a contiguous block of memory. The fragmentation into smaller chunks by the Operating System's Memory Management is to fulfill existing and new memory requests at the time the image was created.


010 Editor - D:\School\Thesis\Results\Set1\control

Open Files
- D:\School\Thesis\Results\Set1\control
- D:\School\Thesis\Results\Set1\test

```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
5C:9F00h: 84 82 01 01 00 FF B5 FC FD FF FF 8B 03 56 FF B5  „‚...ÿµüýÿÿ‹.Vÿµ
5C:9F10h: 1C FF FF FF 8B CB FF 90 6C 01 00 00 33 C9 39 8D  .ÿÿÿ‹Ëÿ.l...3É9.
5C:9F20h: 14 FE FF FF 0F 85 E3 16 00 00 39 8D 78 FE FF FF  .þÿÿ.…ã...9.xþÿÿ
5C:9F30h: 8B 46 04 8B 3E 89 85 68 FE FF FF 0F 85 A7 76 FF  ‹F.‹>‰…hþÿÿ.…§vÿ
5C:9F40h: FF 8B 85 00 FF FF FF 8D 8D 64 FE FF FF 51 FF B5  ÿ‹….ÿÿÿ..dþÿÿQÿµ
5C:9F50h: E4 FE FF FF 89 85 3C FE FF FF FF B5 0C FF FF FF  äþÿÿ‰…<þÿÿÿµ.ÿÿÿ
5C:9F60h: 8B 85 04 FF FF FF 89 85 40 FE FF FF 8B 85 C0 FE  ‹….ÿÿÿ‰…@þÿÿ‹…Àþ
5C:9F70h: FF FF 56 8D 8D 3C FE FF FF 51 8B 8D 1C FF FF FF  ÿÿV..<þÿÿQ‹..ÿÿÿ
5C:9F80h: FF 71 28 89 85 64 FE FF FF 8B 85 AC FE FF FF 89  ÿq(‰…dþÿÿ‹…¬þÿÿ‰
5C:9F90h: 85 68 FE FF FF 8B 03 8B CB FF 90 78 01 00 00 89  …hþÿÿ‹.‹Ëÿ.x...‰
5C:9FA0h: 85 F4 FD FF FF 8B 06 3B F8 0F 85 92 C6 01 00 33  …ôýÿÿ‹.;ø.…’Æ..3
5C:9FB0h: FF 39 BD D8 FE FF FF 0F 84 7A E9 F9 FF 39 BD A4  ÿ9½Øþÿÿ.„zéùÿ9½¤
```

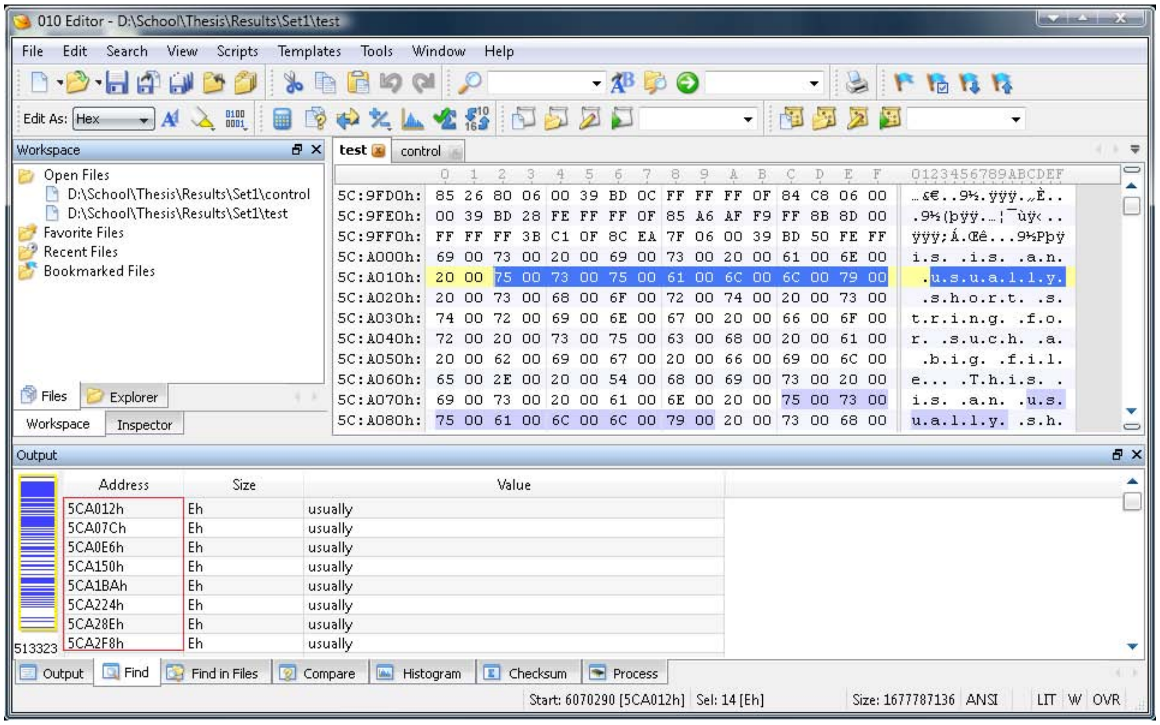
Output

Address	Size	Value
5CA012h	Eh	usually
5CA07Ch	Eh	usually
5CA0E6h	Eh	usually
5CA150h	Eh	usually
5CA1BAh	Eh	usually
5CA224h	Eh	usually
5CA28Eh	Eh	usually
5CA2F8h	Eh	usually

506018

Start: 6070290 [5CA012h] Sel: 14 [Eh] Size: 2136498176 ANSI LIT W OVR

Figure 9. A find on the signature “usually” for set 1 control image


010 Editor - D:\School\Thesis\Results\Set1\test

Open Files
- D:\School\Thesis\Results\Set1\control
- D:\School\Thesis\Results\Set1\test

```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
5C:9FD0h: 85 26 80 06 00 39 BD 0C FF FF FF 0F 84 C8 06 00  …&€..9½.ÿÿÿ.„È..
5C:9FE0h: 00 39 BD 28 FE FF FF 0F 85 A6 AF F9 FF 8B 8D 00  .9½(þÿÿ.…¦¯ùÿ‹..
5C:9FF0h: FF FF FF 3B C1 0F 8C EA 7F 06 00 39 BD 50 FE FF  ÿÿÿ;Á.Œê...9½Pþÿ
5C:A000h: 69 00 73 00 20 00 69 00 73 00 20 00 61 00 6E 00  i.s. .i.s. .a.n.
5C:A010h: 20 00 75 00 73 00 75 00 61 00 6C 00 6C 00 79 00   .u.s.u.a.l.l.y.
5C:A020h: 20 00 73 00 68 00 6F 00 72 00 74 00 20 00 73 00   .s.h.o.r.t. .s.
5C:A030h: 74 00 72 00 69 00 6E 00 67 00 20 00 66 00 6F 00  t.r.i.n.g. .f.o.
5C:A040h: 72 00 20 00 73 00 75 00 63 00 68 00 20 00 61 00  r. .s.u.c.h. .a.
5C:A050h: 20 00 62 00 69 00 67 00 20 00 66 00 69 00 6C 00   .b.i.g. .f.i.l.
5C:A060h: 65 00 2E 00 20 00 54 00 68 00 69 00 73 00 20 00  e... .T.h.i.s. .
5C:A070h: 69 00 73 00 20 00 61 00 6E 00 20 00 75 00 73 00  i.s. .a.n. .u.s.
5C:A080h: 75 00 61 00 6C 00 6C 00 79 00 20 00 73 00 68 00  u.a.l.l.y. .s.h.
```

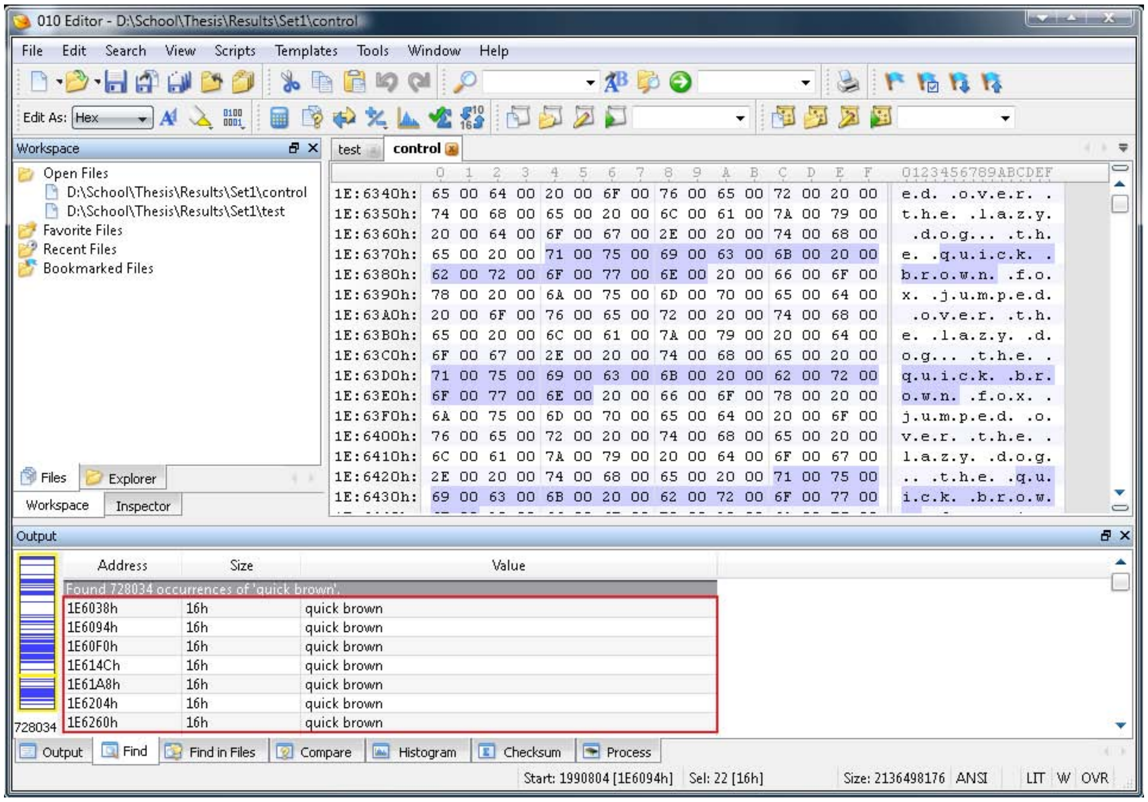
Output

Address	Size	Value
5CA012h	Eh	usually
5CA07Ch	Eh	usually
5CA0E6h	Eh	usually
5CA150h	Eh	usually
5CA1BAh	Eh	usually
5CA224h	Eh	usually
5CA28Eh	Eh	usually
5CA2F8h	Eh	usually

513323

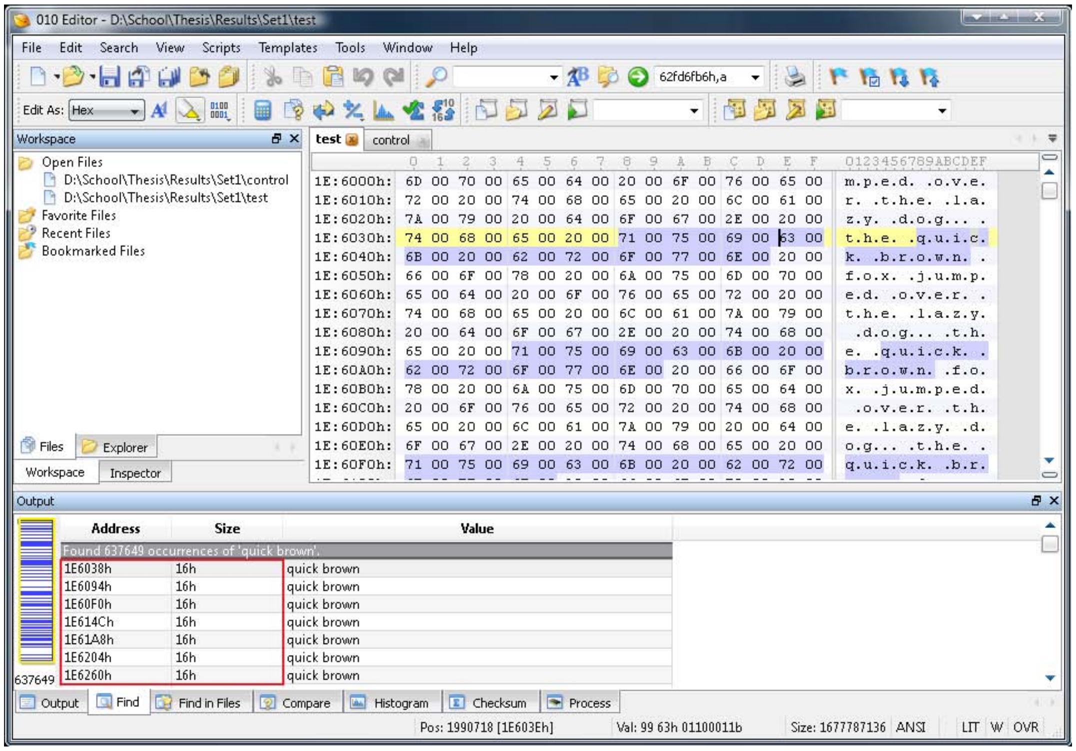
Start: 6070290 [5CA012h] Sel: 14 [Eh] Size: 1677787136 ANSI LIT W OVR

Figure 10. A find on the signature “usually” for set 1 test image



```
010 Editor - D:\School\Thesis\Results\Set1\control

Open Files
  D:\School\Thesis\Results\Set1\control
  D:\School\Thesis\Results\Set1\test

          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
1E:6340h: 65 00 64 00 20 00 6F 00 76 00 65 00 72 00 20 00  e.d. .o.v.e.r. .
1E:6350h: 74 00 68 00 65 00 20 00 6C 00 61 00 7A 00 79 00  t.h.e. .l.a.z.y.
1E:6360h: 20 00 64 00 6F 00 67 00 2E 00 20 00 74 00 68 00   .d.o.g... .t.h.
1E:6370h: 65 00 20 00 71 00 75 00 69 00 63 00 6B 00 20 00  e. .q.u.i.c.k. .
1E:6380h: 62 00 72 00 6F 00 77 00 6E 00 20 00 66 00 6F 00  b.r.o.w.n. .f.o.
1E:6390h: 78 00 20 00 6A 00 75 00 6D 00 70 00 65 00 64 00  x. .j.u.m.p.e.d.
1E:63A0h: 20 00 6F 00 76 00 65 00 72 00 20 00 74 00 68 00   .o.v.e.r. .t.h.
1E:63B0h: 65 00 20 00 6C 00 61 00 7A 00 79 00 20 00 64 00  e. .l.a.z.y. .d.
1E:63C0h: 6F 00 67 00 2E 00 20 00 74 00 68 00 65 00 20 00  o.g... .t.h.e. .
1E:63D0h: 71 00 75 00 69 00 63 00 6B 00 20 00 62 00 72 00  q.u.i.c.k. .b.r.
1E:63E0h: 6F 00 77 00 6E 00 20 00 66 00 6F 00 78 00 20 00  o.w.n. .f.o.x. .
1E:63F0h: 6A 00 75 00 6D 00 70 00 65 00 64 00 20 00 6F 00  j.u.m.p.e.d. .o.
1E:6400h: 76 00 65 00 72 00 20 00 74 00 68 00 65 00 20 00  v.e.r. .t.h.e. .
1E:6410h: 6C 00 61 00 7A 00 79 00 20 00 64 00 6F 00 67 00  l.a.z.y. .d.o.g.
1E:6420h: 2E 00 20 00 74 00 68 00 65 00 20 00 71 00 75 00  .. .t.h.e. .q.u.
1E:6430h: 69 00 63 00 6B 00 20 00 62 00 72 00 6F 00 77 00  i.c.k. .b.r.o.w.

Output
        Address    Size   Value
        Found 728034 occurrences of 'quick brown'.
        1E6038h    16h    quick brown
        1E6094h    16h    quick brown
        1E60F0h    16h    quick brown
        1E614Ch    16h    quick brown
        1E61A8h    16h    quick brown
        1E6204h    16h    quick brown
728034  1E6260h    16h    quick brown

Start: 1990804 [1E6094h]   Sel: 22 [16h]   Size: 2136498176   ANSI   LIT  W  OVR
```

Figure 11. A find on the signature “quick brown” for set 1 control image



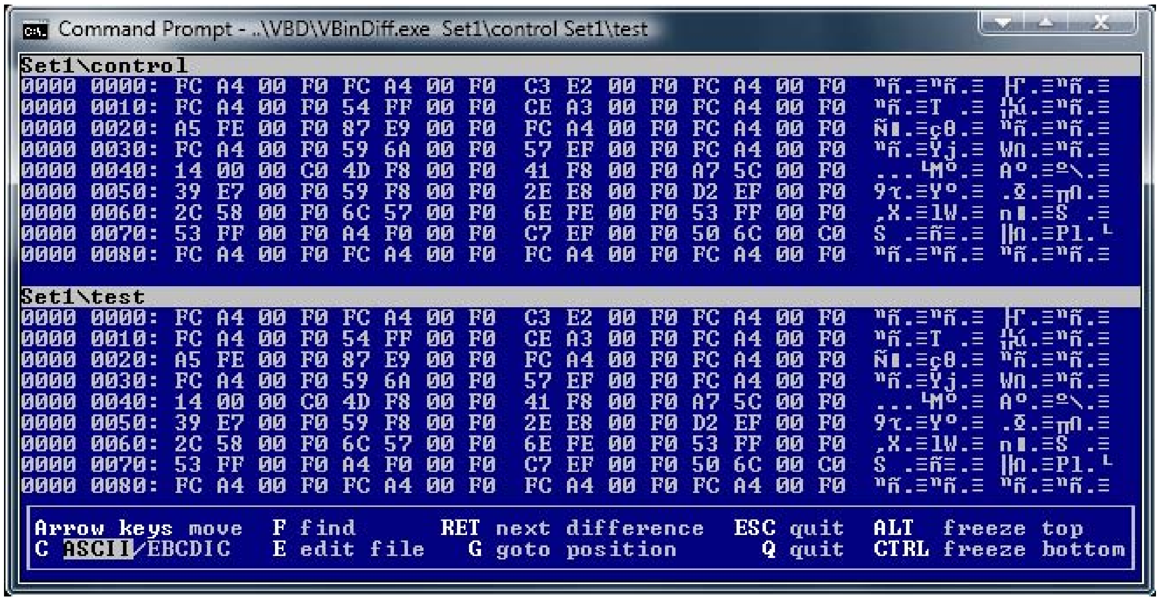
```
010 Editor - D:\School\Thesis\Results\Set1\test

Open Files
  D:\School\Thesis\Results\Set1\control
  D:\School\Thesis\Results\Set1\test

          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
1E:6000h: 6D 00 70 00 65 00 64 00 20 00 6F 00 76 00 65 00  m.p.e.d. .o.v.e.
1E:6010h: 72 00 20 00 74 00 68 00 65 00 20 00 6C 00 61 00  r. .t.h.e. .l.a.
1E:6020h: 7A 00 79 00 20 00 64 00 6F 00 67 00 2E 00 20 00  z.y. .d.o.g... .
1E:6030h: 74 00 68 00 65 00 20 00 71 00 75 00 69 00 63 00  t.h.e. .q.u.i.c.
1E:6040h: 6B 00 20 00 62 00 72 00 6F 00 77 00 6E 00 20 00  k. .b.r.o.w.n. .
1E:6050h: 66 00 6F 00 78 00 20 00 6A 00 75 00 6D 00 70 00  f.o.x. .j.u.m.p.
1E:6060h: 65 00 64 00 20 00 6F 00 76 00 65 00 72 00 20 00  e.d. .o.v.e.r. .
1E:6070h: 74 00 68 00 65 00 20 00 6C 00 61 00 7A 00 79 00  t.h.e. .l.a.z.y.
1E:6080h: 20 00 64 00 6F 00 67 00 2E 00 20 00 74 00 68 00   .d.o.g... .t.h.
1E:6090h: 65 00 20 00 71 00 75 00 69 00 63 00 6B 00 20 00  e. .q.u.i.c.k. .
1E:60A0h: 62 00 72 00 6F 00 77 00 6E 00 20 00 66 00 6F 00  b.r.o.w.n. .f.o.
1E:60B0h: 78 00 20 00 6A 00 75 00 6D 00 70 00 65 00 64 00  x. .j.u.m.p.e.d.
1E:60C0h: 20 00 6F 00 76 00 65 00 72 00 20 00 74 00 68 00   .o.v.e.r. .t.h.
1E:60D0h: 65 00 20 00 6C 00 61 00 7A 00 79 00 20 00 64 00  e. .l.a.z.y. .d.
1E:60E0h: 6F 00 67 00 2E 00 20 00 74 00 68 00 65 00 20 00  o.g... .t.h.e. .
1E:60F0h: 71 00 75 00 69 00 63 00 6B 00 20 00 62 00 72 00  q.u.i.c.k. .b.r.

Output
        Address    Size   Value
        Found 637649 occurrences of 'quick brown'.
        1E6038h    16h    quick brown
        1E6094h    16h    quick brown
        1E60F0h    16h    quick brown
        1E614Ch    16h    quick brown
        1E61A8h    16h    quick brown
        1E6204h    16h    quick brown
637649  1E6260h    16h    quick brown

Pos: 1990718 [1E603Eh]   Val: 99 63h 01100011b   Size: 1677787136   ANSI   LIT  W  OVR
```

Figure 12. A find on the signature “quick brown” for set 1 test image

Both the signatures are littered throughout the entire length of the control image. These are the blue segments as on Figure 10 and Figure 12. Since the test image captured is incomplete (approximately 440MB less), an approximation of it would be the first 1600 MB of the control image. The Memory Management Unit of the Operating System would have made some changes to memory in between the time the two images are captured, but the general spread pattern should still be observable. The vertical yellow line in Figure 12 marks the end of the capture test image. The top half of the boxed region would be an approximate of the captured test image. Compare this to the boxed region in Figure 13. The blue regions are more spread out, imagine stretching the box region in Figure 12, but the similarity between the two is apparent. This is observable from Figure 10 and 11 as well.

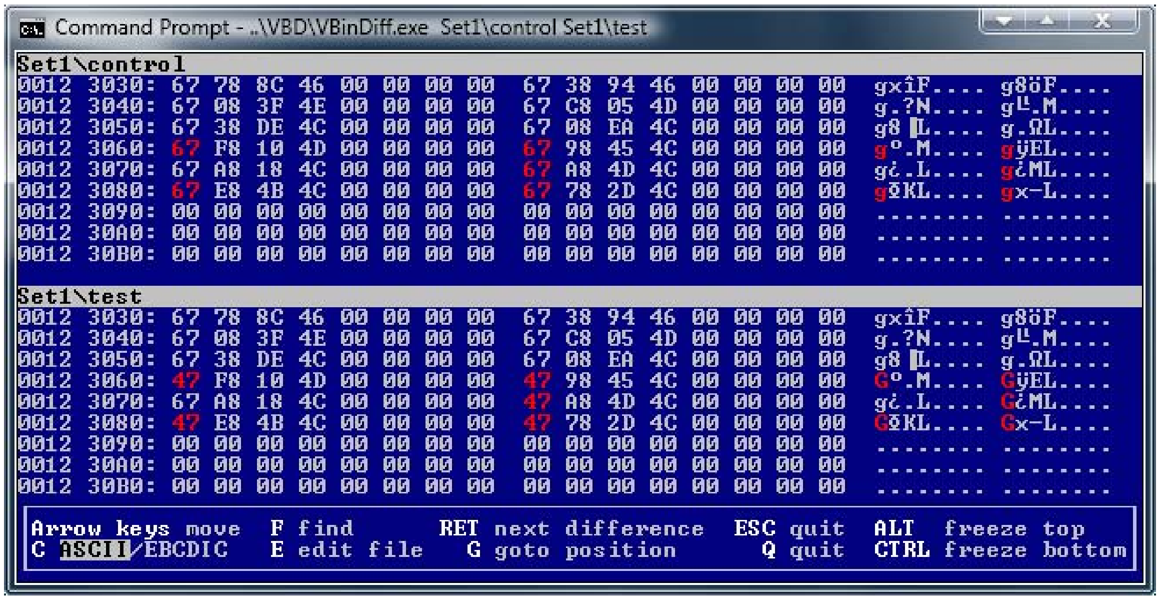
3. Testing by Comparison

The VBinDiff tool was used to compare the control and test image of each set. Figure 13 show that the images in set 1 are largely the same at the beginning of the images. The first difference between the two images is at the offset position 0x00123060 as seen in Figure 14. The two images remain largely the same with slight difference, like those in Figure 14, until the offset position 0x0014C000. These changes can be due to the memory requirement needed by the MemReader application and driver. From position 0x0014C000 onwards, the two images start to differ. Figure 15 shows the image at position 0x0014C000.



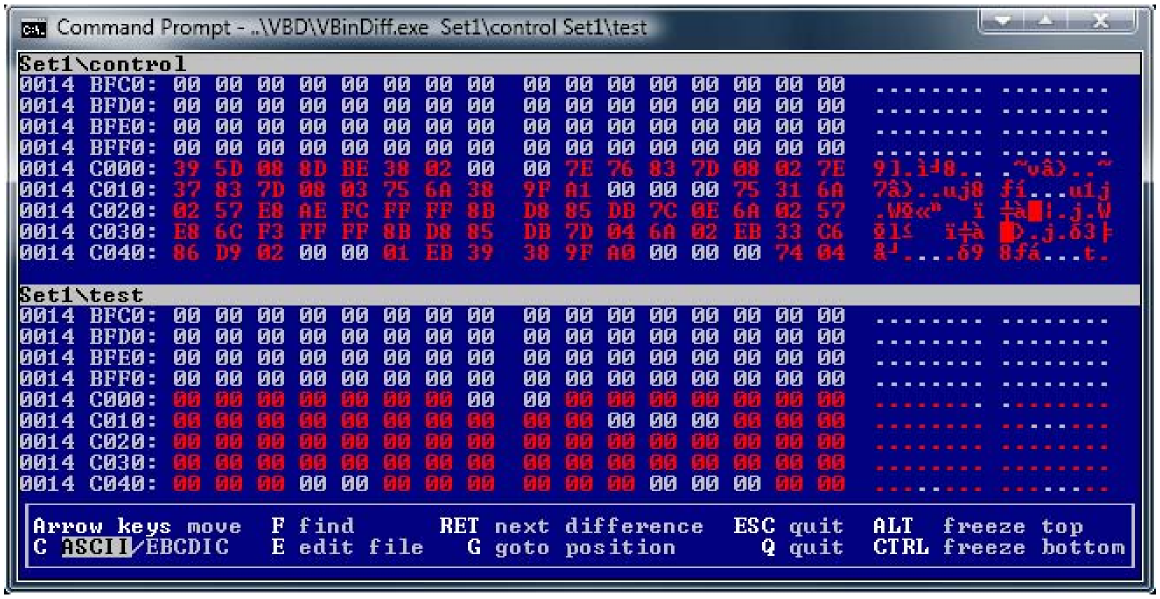
```
Command Prompt - ..\VBD\VBinDiff.exe  Set1\control Set1\test
Set1\control
0000 0000: FC A4 00 F0 FC A4 00 F0  C3 E2 00 F0 FC A4 00 F0  ⁿñ.≡ⁿñ.≡ ├Γ.≡ⁿñ.≡
0000 0010: FC A4 00 F0 54 FF 00 F0  CE A3 00 F0 FC A4 00 F0  ⁿñ.≡T .≡ ╬ú.≡ⁿñ.≡
0000 0020: A5 FE 00 F0 87 E9 00 F0  FC A4 00 F0 FC A4 00 F0  Ñ■.≡çΘ.≡ ⁿñ.≡ⁿñ.≡
0000 0030: FC A4 00 F0 59 6A 00 F0  57 EF 00 F0 FC A4 00 F0  ⁿñ.≡Yj.≡ W∩.≡ⁿñ.≡
0000 0040: 14 00 00 C0 4D F8 00 F0  41 F8 00 F0 A7 5C 00 F0  ...└M°.≡ A°.≡º\.≡
0000 0050: 39 E7 00 F0 59 F8 00 F0  2E E8 00 F0 D2 EF 00 F0  9τ.≡Y°.≡ .Φ.≡╥∩.≡
0000 0060: 2C 58 00 F0 6C 57 00 F0  6E FE 00 F0 53 FF 00 F0  ,X.≡lW.≡ n■.≡S .≡
0000 0070: 53 FF 00 F0 A4 F0 00 F0  C7 EF 00 F0 50 6C 00 C0  S .≡ñ≡.≡ ╟∩.≡Pl.└
0000 0080: FC A4 00 F0 FC A4 00 F0  FC A4 00 F0 FC A4 00 F0  ⁿñ.≡ⁿñ.≡ ⁿñ.≡ⁿñ.≡

Set1\test
0000 0000: FC A4 00 F0 FC A4 00 F0  C3 E2 00 F0 FC A4 00 F0  ⁿñ.≡ⁿñ.≡ ├Γ.≡ⁿñ.≡
0000 0010: FC A4 00 F0 54 FF 00 F0  CE A3 00 F0 FC A4 00 F0  ⁿñ.≡T .≡ ╬ú.≡ⁿñ.≡
0000 0020: A5 FE 00 F0 87 E9 00 F0  FC A4 00 F0 FC A4 00 F0  Ñ■.≡çΘ.≡ ⁿñ.≡ⁿñ.≡
0000 0030: FC A4 00 F0 59 6A 00 F0  57 EF 00 F0 FC A4 00 F0  ⁿñ.≡Yj.≡ W∩.≡ⁿñ.≡
0000 0040: 14 00 00 C0 4D F8 00 F0  41 F8 00 F0 A7 5C 00 F0  ...└M°.≡ A°.≡º\.≡
0000 0050: 39 E7 00 F0 59 F8 00 F0  2E E8 00 F0 D2 EF 00 F0  9τ.≡Y°.≡ .Φ.≡╥∩.≡
0000 0060: 2C 58 00 F0 6C 57 00 F0  6E FE 00 F0 53 FF 00 F0  ,X.≡lW.≡ n■.≡S .≡
0000 0070: 53 FF 00 F0 A4 F0 00 F0  C7 EF 00 F0 50 6C 00 C0  S .≡ñ≡.≡ ╟∩.≡Pl.└
0000 0080: FC A4 00 F0 FC A4 00 F0  FC A4 00 F0 FC A4 00 F0  ⁿñ.≡ⁿñ.≡ ⁿñ.≡ⁿñ.≡

 Arrow keys move  F find      RET next difference  ESC quit  ALT  freeze top
 C ASCII/EBCDIC   E edit file   G goto position      Q quit  CTRL freeze bottom
```

Figure 13. Set 1: No difference between the images (till 0x00123050)



```
Command Prompt - ..\VBD\VBinDiff.exe  Set1\control Set1\test
Set1\control
0012 3030: 67 78 8C 46 00 00 00 00  67 38 94 46 00 00 00 00  gxîF.... g8öF....
0012 3040: 67 08 3F 4E 00 00 00 00  67 C8 05 4D 00 00 00 00  g.?N.... g╚.M....
0012 3050: 67 38 DE 4C 00 00 00 00  67 08 EA 4C 00 00 00 00  g8▐L.... g.ΩL....
0012 3060: 67 F8 10 4D 00 00 00 00  67 98 45 4C 00 00 00 00  g°.M.... gÿEL....
0012 3070: 67 A8 18 4C 00 00 00 00  67 A8 4D 4C 00 00 00 00  g¿.L.... g¿ML....
0012 3080: 67 E8 4B 4C 00 00 00 00  67 78 2D 4C 00 00 00 00  gΦKL.... gx-L....
0012 3090: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0012 30A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0012 30B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........

Set1\test
0012 3030: 67 78 8C 46 00 00 00 00  67 38 94 46 00 00 00 00  gxîF.... g8öF....
0012 3040: 67 08 3F 4E 00 00 00 00  67 C8 05 4D 00 00 00 00  g.?N.... g╚.M....
0012 3050: 67 38 DE 4C 00 00 00 00  67 08 EA 4C 00 00 00 00  g8▐L.... g.ΩL....
0012 3060: 47 F8 10 4D 00 00 00 00  47 98 45 4C 00 00 00 00  G°.M.... GÿEL....
0012 3070: 67 A8 18 4C 00 00 00 00  47 A8 4D 4C 00 00 00 00  g¿.L.... G¿ML....
0012 3080: 47 E8 4B 4C 00 00 00 00  47 78 2D 4C 00 00 00 00  GΦKL.... Gx-L....
0012 3090: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0012 30A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0012 30B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........

 Arrow keys move  F find      RET next difference  ESC quit  ALT  freeze top
 C ASCII/EBCDIC   E edit file   G goto position      Q quit  CTRL freeze bottom
```

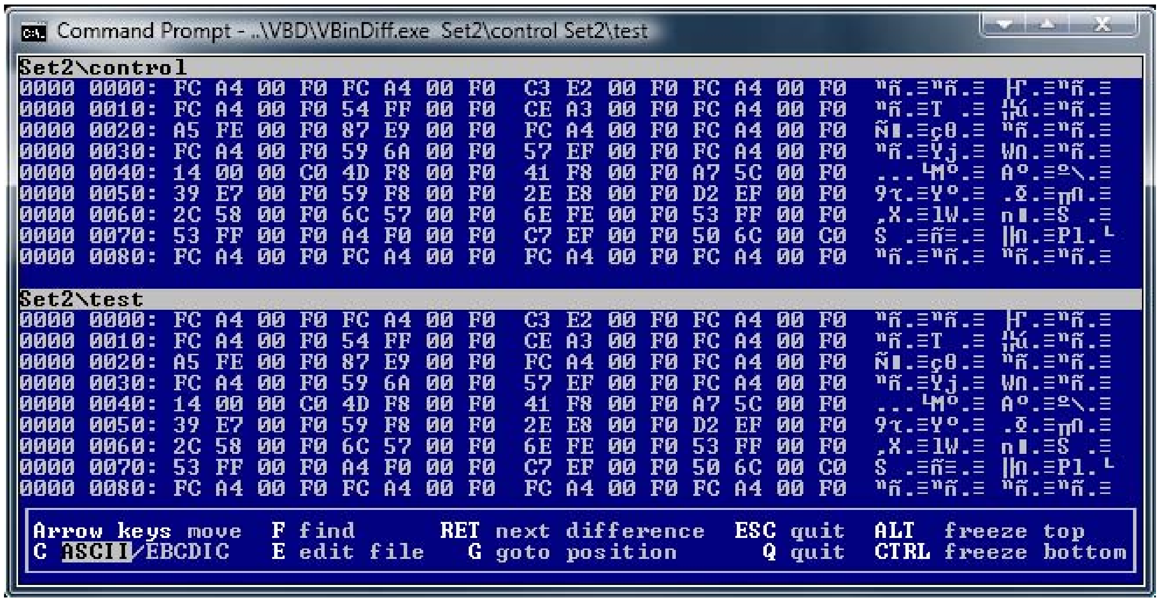
Figure 14. Set 1: First difference between the images (0x00123060)



```
Command Prompt - ..\VBD\VBinDiff.exe  Set1\control Set1\test
Set1\control
0014 BFC0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0014 BFD0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0014 BFE0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0014 BFF0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0014 C000: 39 5D 08 8D BE 38 02 00  00 7E 76 83 7D 08 02 7E  9].ì╛8.. .~vâ}..~
0014 C010: 37 83 7D 08 03 75 6A 38  9F A1 00 00 00 75 31 6A  7â}..uj8 ƒí...u1j
0014 C020: 02 57 E8 AE FC FF FF 8B  D8 85 DB 7C 0E 6A 02 57  .WΦ«ⁿ  ï ╪à█|.j.W
0014 C030: E8 6C F3 FF FF 8B D8 85  DB 7D 04 6A 02 EB 33 C6  Φl≤  ï╪à █}.j.δ3╞
0014 C040: 86 D9 02 00 00 01 EB 39  38 9F A0 00 00 00 74 04  å┘....δ9 8ƒá...t.

Set1\test
0014 BFC0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0014 BFD0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0014 BFE0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0014 BFF0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0014 C000: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0014 C010: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0014 C020: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0014 C030: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0014 C040: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........

 Arrow keys move   F find      RET next difference   ESC quit   ALT  freeze top
 C ASCII/EBCDIC    E edit file   G goto position       Q quit   CTRL freeze bottom
```

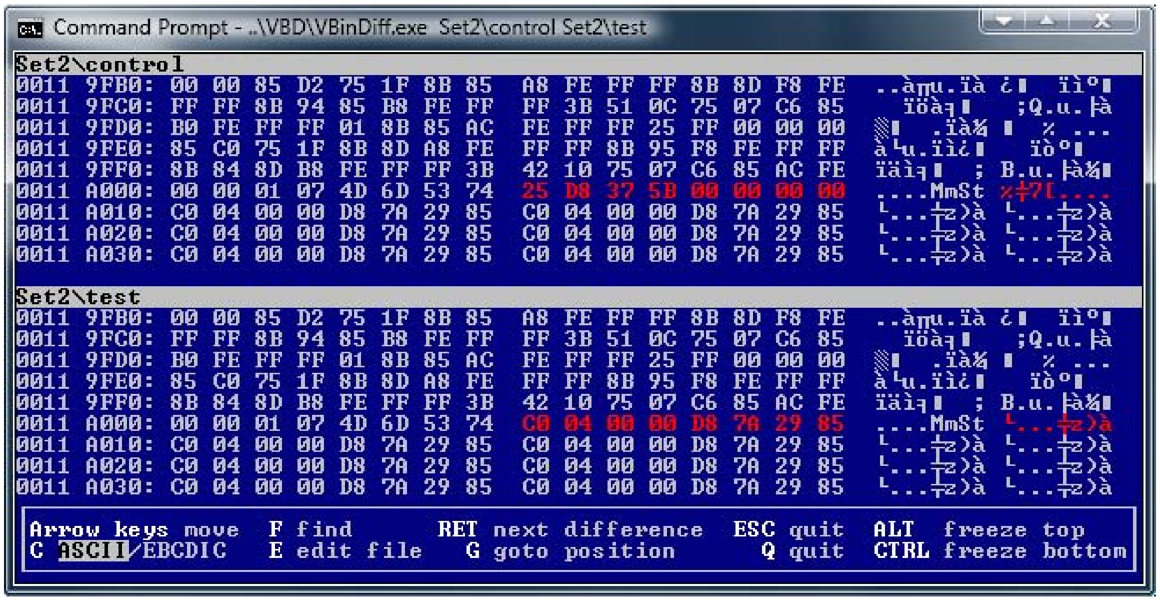
Figure 15. Set 1: Position (0x0014C000) where images start to differ

The result is consistent with the images extracted in set 2, see Figure 16 – 18, and set 3, see Figure 19 – 21. The only notable difference is that point which the data on the two images start to get drastically different. In both set 2 and set 3, the tripping point is at 0x0015F000 – 76KB later than set 1.



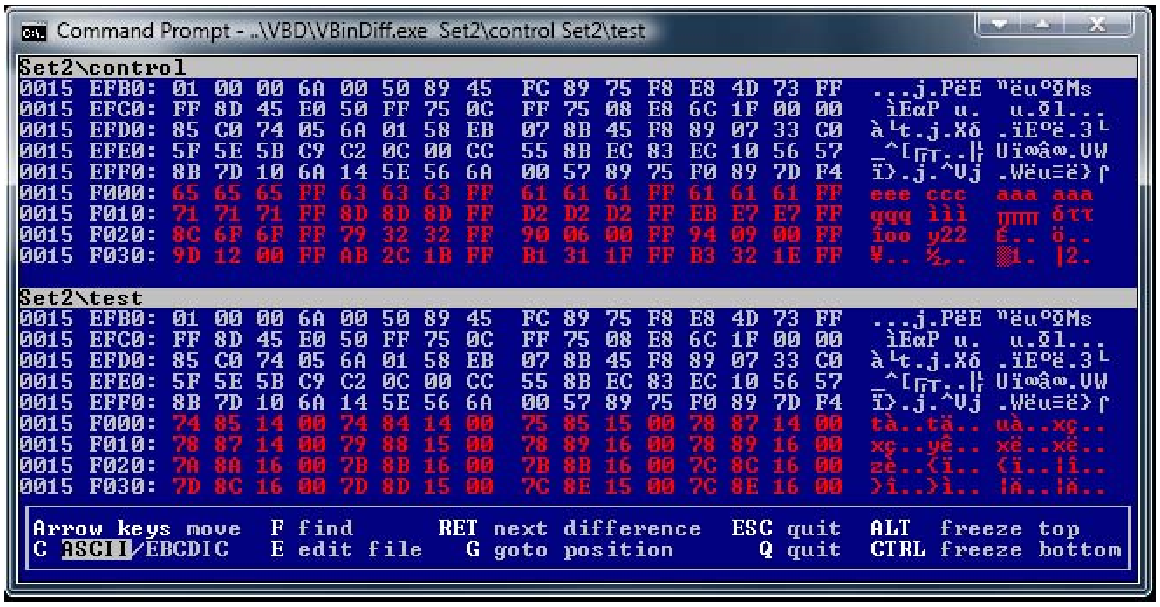
```
Command Prompt - ..\VBD\VBinDiff.exe  Set2\control Set2\test
Set2\control
0000 0000: FC A4 00 F0 FC A4 00 F0  C3 E2 00 F0 FC A4 00 F0  ⁿñ.≡ⁿñ.≡ ├Γ.≡ⁿñ.≡
0000 0010: FC A4 00 F0 54 FF 00 F0  CE A3 00 F0 FC A4 00 F0  ⁿñ.≡T .≡ ╬ú.≡ⁿñ.≡
0000 0020: A5 FE 00 F0 87 E9 00 F0  FC A4 00 F0 FC A4 00 F0  Ñ■.≡çΘ.≡ ⁿñ.≡ⁿñ.≡
0000 0030: FC A4 00 F0 59 6A 00 F0  57 EF 00 F0 FC A4 00 F0  ⁿñ.≡Yj.≡ Wn.≡ⁿñ.≡
0000 0040: 14 00 00 C0 4D F8 00 F0  41 F8 00 F0 A7 5C 00 F0  ...└M°.≡ A°.≡º\.≡
0000 0050: 39 E7 00 F0 59 F8 00 F0  2E E8 00 F0 D2 EF 00 F0  9τ.≡Y°.≡ .Φ.≡╥n.≡
0000 0060: 2C 58 00 F0 6C 57 00 F0  6E FE 00 F0 53 FF 00 F0  ,X.≡lW.≡ n■.≡S .≡
0000 0070: 53 FF 00 F0 A4 F0 00 F0  C7 EF 00 F0 50 6C 00 C0  S .≡ñ≡.≡ ╟n.≡Pl.└
0000 0080: FC A4 00 F0 FC A4 00 F0  FC A4 00 F0 FC A4 00 F0  ⁿñ.≡ⁿñ.≡ ⁿñ.≡ⁿñ.≡

Set2\test
0000 0000: FC A4 00 F0 FC A4 00 F0  C3 E2 00 F0 FC A4 00 F0  ⁿñ.≡ⁿñ.≡ ├Γ.≡ⁿñ.≡
0000 0010: FC A4 00 F0 54 FF 00 F0  CE A3 00 F0 FC A4 00 F0  ⁿñ.≡T .≡ ╬ú.≡ⁿñ.≡
0000 0020: A5 FE 00 F0 87 E9 00 F0  FC A4 00 F0 FC A4 00 F0  Ñ■.≡çΘ.≡ ⁿñ.≡ⁿñ.≡
0000 0030: FC A4 00 F0 59 6A 00 F0  57 EF 00 F0 FC A4 00 F0  ⁿñ.≡Yj.≡ Wn.≡ⁿñ.≡
0000 0040: 14 00 00 C0 4D F8 00 F0  41 F8 00 F0 A7 5C 00 F0  ...└M°.≡ A°.≡º\.≡
0000 0050: 39 E7 00 F0 59 F8 00 F0  2E E8 00 F0 D2 EF 00 F0  9τ.≡Y°.≡ .Φ.≡╥n.≡
0000 0060: 2C 58 00 F0 6C 57 00 F0  6E FE 00 F0 53 FF 00 F0  ,X.≡lW.≡ n■.≡S .≡
0000 0070: 53 FF 00 F0 A4 F0 00 F0  C7 EF 00 F0 50 6C 00 C0  S .≡ñ≡.≡ ╟n.≡Pl.└
0000 0080: FC A4 00 F0 FC A4 00 F0  FC A4 00 F0 FC A4 00 F0  ⁿñ.≡ⁿñ.≡ ⁿñ.≡ⁿñ.≡

 Arrow keys move   F find      RET next difference   ESC quit   ALT  freeze top
 C ASCII/EBCDIC    E edit file   G goto position       Q quit   CTRL freeze bottom
```

Figure 16. Set 2: No difference between the images (till 0x0011A000)



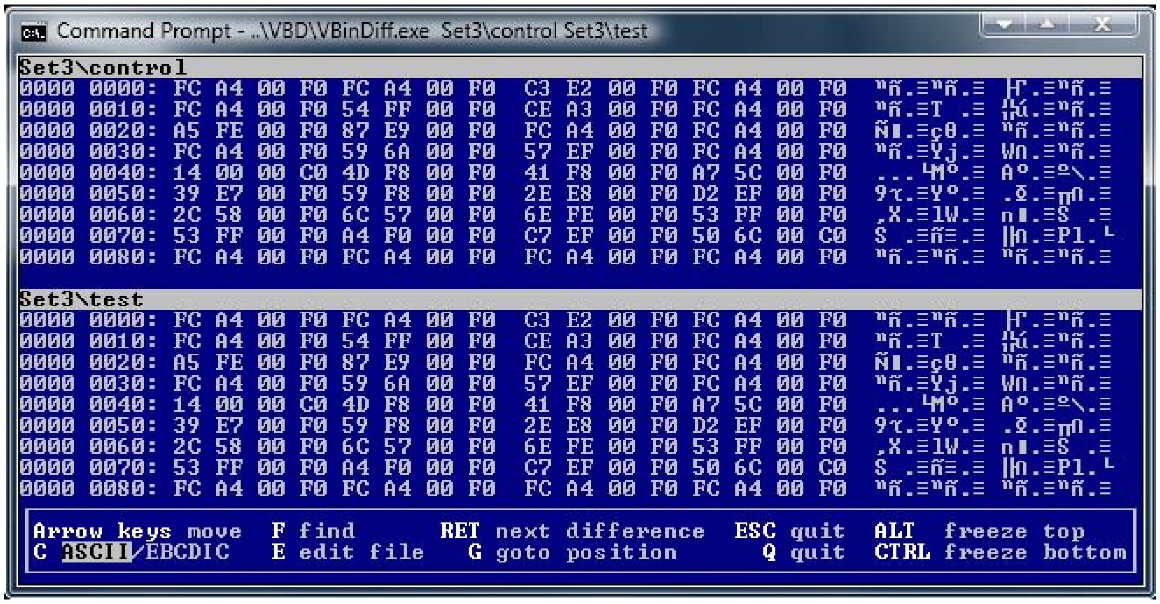
```
Command Prompt - ..\VBD\VBinDiff.exe  Set2\control Set2\test
Set2\control
0011 9FB0: 00 00 85 D2 75 1F 8B 85  A8 FE FF FF 8B 8D F8 FE  ..àπu.ïà ¿▮  ïì°▮
0011 9FC0: FF FF 8B 94 85 B8 FE FF  FF 3B 51 0C 75 07 C6 85    ïöà╕▮    ;Q.u.╞à
0011 9FD0: B0 FE FF FF 01 8B 85 AC  FE FF FF 25 FF 00 00 00  ░▮   .ïà¼ ▮  % ...
0011 9FE0: 85 C0 75 1F 8B 8D A8 FE  FF FF 8B 95 F8 FE FF FF  à└u.ïì¿▮    ïò°▮
0011 9FF0: 8B 84 8D B8 FE FF FF 3B  42 10 75 07 C6 85 AC FE  ïäì╕▮   ; B.u.╞à¼▮
0011 A000: 00 00 01 07 4D 6D 53 74  25 D8 37 5B 00 00 00 00  ....MmSt %╪7[....
0011 A010: C0 04 00 00 D8 7A 29 85  C0 04 00 00 D8 7A 29 85  └...╪z)à └...╪z)à
0011 A020: C0 04 00 00 D8 7A 29 85  C0 04 00 00 D8 7A 29 85  └...╪z)à └...╪z)à
0011 A030: C0 04 00 00 D8 7A 29 85  C0 04 00 00 D8 7A 29 85  └...╪z)à └...╪z)à

Set2\test
0011 9FB0: 00 00 85 D2 75 1F 8B 85  A8 FE FF FF 8B 8D F8 FE  ..àπu.ïà ¿▮  ïì°▮
0011 9FC0: FF FF 8B 94 85 B8 FE FF  FF 3B 51 0C 75 07 C6 85    ïöà╕▮    ;Q.u.╞à
0011 9FD0: B0 FE FF FF 01 8B 85 AC  FE FF FF 25 FF 00 00 00  ░▮   .ïà¼ ▮  % ...
0011 9FE0: 85 C0 75 1F 8B 8D A8 FE  FF FF 8B 95 F8 FE FF FF  à└u.ïì¿▮    ïò°▮
0011 9FF0: 8B 84 8D B8 FE FF FF 3B  42 10 75 07 C6 85 AC FE  ïäì╕▮   ; B.u.╞à¼▮
0011 A000: 00 00 01 07 4D 6D 53 74  C0 04 00 00 D8 7A 29 85  ....MmSt └...╪z)à
0011 A010: C0 04 00 00 D8 7A 29 85  C0 04 00 00 D8 7A 29 85  └...╪z)à └...╪z)à
0011 A020: C0 04 00 00 D8 7A 29 85  C0 04 00 00 D8 7A 29 85  └...╪z)à └...╪z)à
0011 A030: C0 04 00 00 D8 7A 29 85  C0 04 00 00 D8 7A 29 85  └...╪z)à └...╪z)à

 Arrow keys move   F find       RET next difference  ESC quit  ALT  freeze top
 C ASCII/EBCDIC    E edit file    G goto position      Q quit  CTRL freeze bottom
```

Figure 17. Set 2: First difference between the images (0x0011A000)



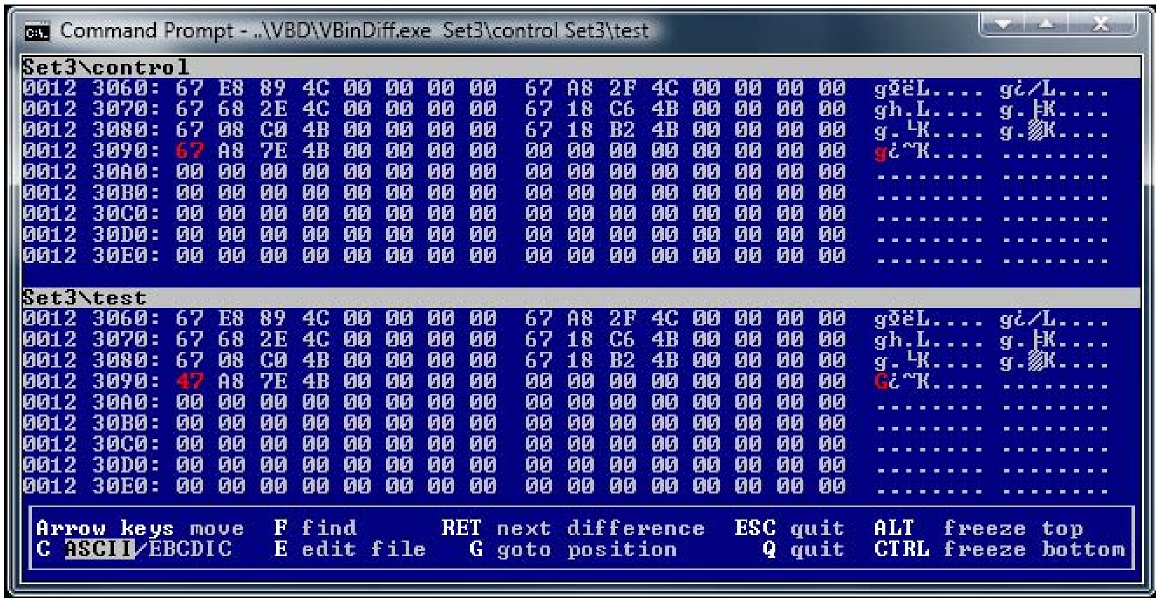
```
Command Prompt - ..\VBD\VBinDiff.exe  Set2\control Set2\test
Set2\control
0015 EFB0: 01 00 00 6A 00 50 89 45  FC 89 75 F8 E8 4D 73 FF  ...j.PëE ⁿëu°ΦMs
0015 EFC0: FF 8D 45 E0 50 FF 75 0C  FF 75 08 E8 6C 1F 00 00   ìEαP u.  u.Φl...
0015 EFD0: 85 C0 74 05 6A 01 58 EB  07 8B 45 F8 89 07 33 C0  à└t.j.Xδ .ïE°ë.3└
0015 EFE0: 5F 5E 5B C9 C2 0C 00 CC  55 8B EC 83 EC 10 56 57  _^[╔┬..╠ Uï∞â∞.VW
0015 EFF0: 8B 7D 10 6A 14 5E 56 6A  00 57 89 75 F0 89 7D F4  ï}.j.^Vj .Wëu≡ë}⌠
0015 F000: 65 65 65 FF 63 63 63 FF  61 61 61 FF 61 61 61 FF  eee ccc  aaa aaa
0015 F010: 71 71 71 FF 8D 8D 8D FF  D2 D2 D2 FF EB E7 E7 FF  qqq ììì  πππ δττ
0015 F020: 8C 6F 6F FF 79 32 32 FF  90 06 00 FF 94 09 00 FF  îoo y22  É.. ö..
0015 F030: 9D 12 00 FF AB 2C 1B FF  B1 31 1F FF B3 32 1E FF  ¥.. ½,.  ▒1. │2.

Set2\test
0015 EFB0: 01 00 00 6A 00 50 89 45  FC 89 75 F8 E8 4D 73 FF  ...j.PëE ⁿëu°ΦMs
0015 EFC0: FF 8D 45 E0 50 FF 75 0C  FF 75 08 E8 6C 1F 00 00   ìEαP u.  u.Φl...
0015 EFD0: 85 C0 74 05 6A 01 58 EB  07 8B 45 F8 89 07 33 C0  à└t.j.Xδ .ïE°ë.3└
0015 EFE0: 5F 5E 5B C9 C2 0C 00 CC  55 8B EC 83 EC 10 56 57  _^[╔┬..╠ Uï∞â∞.VW
0015 EFF0: 8B 7D 10 6A 14 5E 56 6A  00 57 89 75 F0 89 7D F4  ï}.j.^Vj .Wëu≡ë}⌠
0015 F000: 74 85 14 00 74 84 14 00  75 85 15 00 78 87 14 00  tà..tä.. uà..xç..
0015 F010: 78 87 14 00 79 88 15 00  78 89 16 00 78 89 16 00  xç..yê.. xë..xë..
0015 F020: 7A 8A 16 00 7B 8B 16 00  7B 8B 16 00 7C 8C 16 00  zè..{ï.. {ï..|î..
0015 F030: 7D 8C 16 00 7D 8D 15 00  7C 8E 15 00 7C 8E 16 00  }î..}ì.. |Ä..|Ä..

 Arrow keys move   F find       RET next difference  ESC quit  ALT  freeze top
 C ASCII/EBCDIC    E edit file    G goto position      Q quit  CTRL freeze bottom
```

Figure 18. Set 2: Position (0x0015F000) where images start to differ



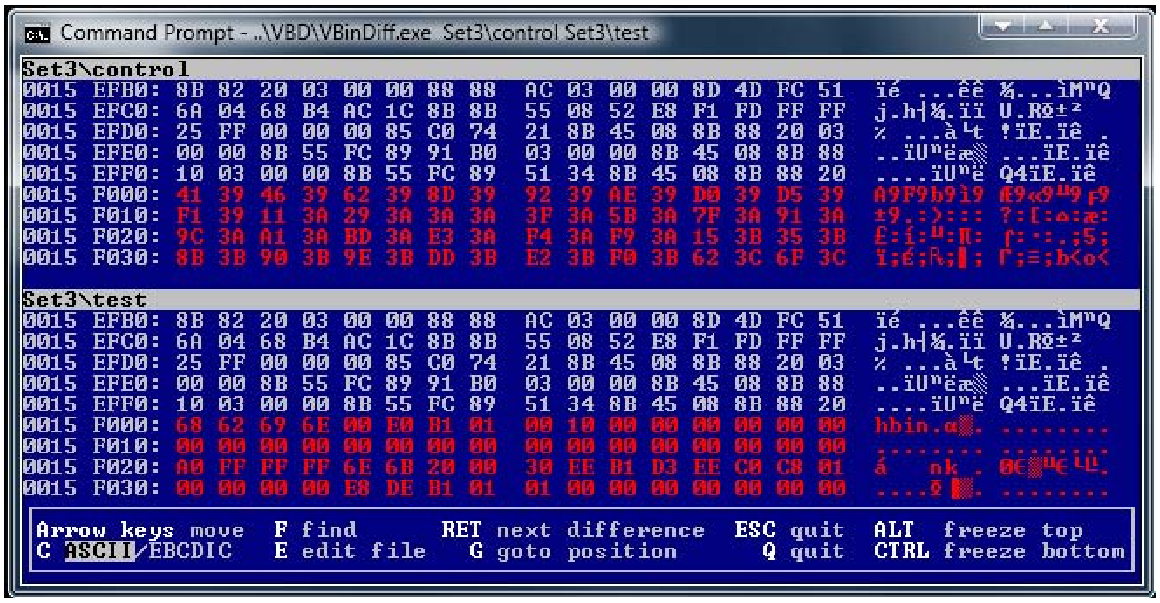
```
Command Prompt - ..\VBD\VBinDiff.exe  Set3\control Set3\test
Set3\control
0000 0000: FC A4 00 F0 FC A4 00 F0  C3 E2 00 F0 FC A4 00 F0  ⁿñ.≡ⁿñ.≡ ├Γ.≡ⁿñ.≡
0000 0010: FC A4 00 F0 54 FF 00 F0  CE A3 00 F0 FC A4 00 F0  ⁿñ.≡T .≡ ╬ú.≡ⁿñ.≡
0000 0020: A5 FE 00 F0 87 E9 00 F0  FC A4 00 F0 FC A4 00 F0  Ñ■.≡çΘ.≡ ⁿñ.≡ⁿñ.≡
0000 0030: FC A4 00 F0 59 6A 00 F0  57 EF 00 F0 FC A4 00 F0  ⁿñ.≡Yj.≡ W∩.≡ⁿñ.≡
0000 0040: 14 00 00 C0 4D F8 00 F0  41 F8 00 F0 A7 5C 00 F0  ...└M°.≡ A°.≡º\.≡
0000 0050: 39 E7 00 F0 59 F8 00 F0  2E E8 00 F0 D2 EF 00 F0  9τ.≡Y°.≡ .Φ.≡╥∩.≡
0000 0060: 2C 58 00 F0 6C 57 00 F0  6E FE 00 F0 53 FF 00 F0  ,X.≡lW.≡ n■.≡S .≡
0000 0070: 53 FF 00 F0 A4 F0 00 F0  C7 EF 00 F0 50 6C 00 C0  S .≡ñ≡.≡ ╟∩.≡Pl.└
0000 0080: FC A4 00 F0 FC A4 00 F0  FC A4 00 F0 FC A4 00 F0  ⁿñ.≡ⁿñ.≡ ⁿñ.≡ⁿñ.≡

Set3\test
0000 0000: FC A4 00 F0 FC A4 00 F0  C3 E2 00 F0 FC A4 00 F0  ⁿñ.≡ⁿñ.≡ ├Γ.≡ⁿñ.≡
0000 0010: FC A4 00 F0 54 FF 00 F0  CE A3 00 F0 FC A4 00 F0  ⁿñ.≡T .≡ ╬ú.≡ⁿñ.≡
0000 0020: A5 FE 00 F0 87 E9 00 F0  FC A4 00 F0 FC A4 00 F0  Ñ■.≡çΘ.≡ ⁿñ.≡ⁿñ.≡
0000 0030: FC A4 00 F0 59 6A 00 F0  57 EF 00 F0 FC A4 00 F0  ⁿñ.≡Yj.≡ W∩.≡ⁿñ.≡
0000 0040: 14 00 00 C0 4D F8 00 F0  41 F8 00 F0 A7 5C 00 F0  ...└M°.≡ A°.≡º\.≡
0000 0050: 39 E7 00 F0 59 F8 00 F0  2E E8 00 F0 D2 EF 00 F0  9τ.≡Y°.≡ .Φ.≡╥∩.≡
0000 0060: 2C 58 00 F0 6C 57 00 F0  6E FE 00 F0 53 FF 00 F0  ,X.≡lW.≡ n■.≡S .≡
0000 0070: 53 FF 00 F0 A4 F0 00 F0  C7 EF 00 F0 50 6C 00 C0  S .≡ñ≡.≡ ╟∩.≡Pl.└
0000 0080: FC A4 00 F0 FC A4 00 F0  FC A4 00 F0 FC A4 00 F0  ⁿñ.≡ⁿñ.≡ ⁿñ.≡ⁿñ.≡

 Arrow keys move  F find      RET next difference  ESC quit  ALT  freeze top
 C ASCII/EBCDIC   E edit file   G goto position      Q quit  CTRL freeze bottom
```

Figure 19. Set 3: No difference between the images (till 0x00123090)



```
Command Prompt - ..\VBD\VBinDiff.exe  Set3\control Set3\test
Set3\control
0012 3060: 67 E8 89 4C 00 00 00 00  67 A8 2F 4C 00 00 00 00  gΦëL.... g¿/L....
0012 3070: 67 68 2E 4C 00 00 00 00  67 18 C6 4B 00 00 00 00  gh.L.... g.╞K....
0012 3080: 67 08 C0 4B 00 00 00 00  67 18 B2 4B 00 00 00 00  g.└K.... g.▓K....
0012 3090: 67 A8 7E 4B 00 00 00 00  00 00 00 00 00 00 00 00  g¿~K.... ........
0012 30A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0012 30B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0012 30C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0012 30D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0012 30E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........

Set3\test
0012 3060: 67 E8 89 4C 00 00 00 00  67 A8 2F 4C 00 00 00 00  gΦëL.... g¿/L....
0012 3070: 67 68 2E 4C 00 00 00 00  67 18 C6 4B 00 00 00 00  gh.L.... g.╞K....
0012 3080: 67 08 C0 4B 00 00 00 00  67 18 B2 4B 00 00 00 00  g.└K.... g.▓K....
0012 3090: 47 A8 7E 4B 00 00 00 00  00 00 00 00 00 00 00 00  G¿~K.... ........
0012 30A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0012 30B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0012 30C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0012 30D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0012 30E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........

 Arrow keys move  F find      RET next difference  ESC quit  ALT  freeze top
 C ASCII/EBCDIC   E edit file   G goto position      Q quit  CTRL freeze bottom
```

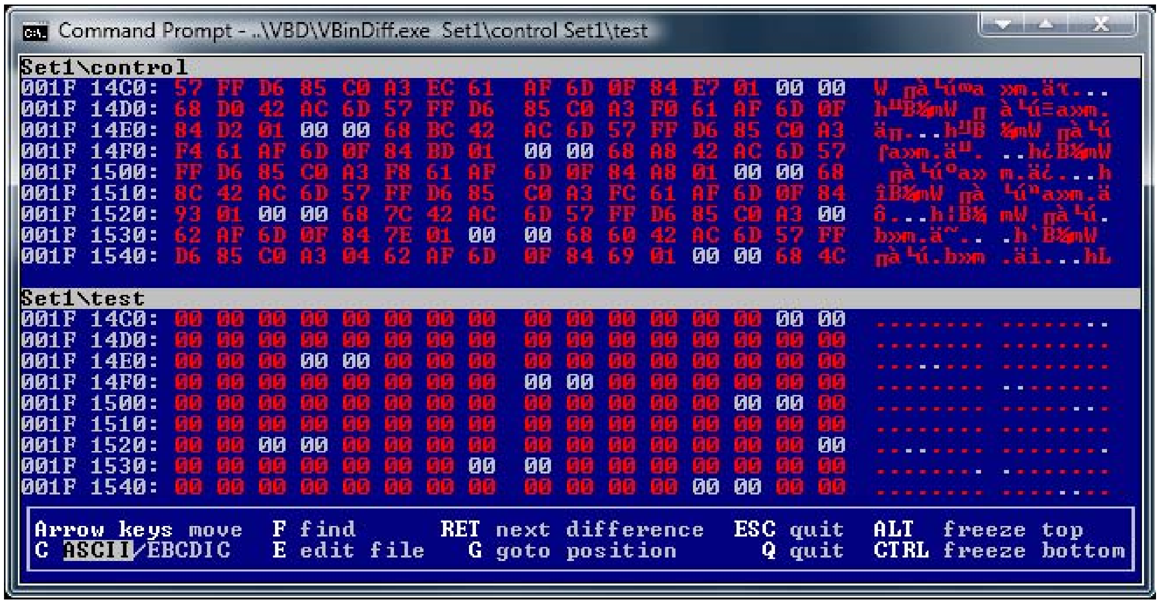
Figure 20. Set 3: First difference between images (0x00123090)



```
Command Prompt - ..\VBD\VBinDiff.exe  Set3\control Set3\test
Set3\control
0015 EFB0: 8B 82 20 03 00 00 88 88  AC 03 00 00 8D 4D FC 51  ïé ...êê ¼...ìM"Q
0015 EFC0: 6A 04 68 B4 AC 1C 8B 8B  55 08 52 E8 F1 FD FF FF  j.h┤¼.ïï U.RΦ±²
0015 EFD0: 25 FF 00 00 00 85 C0 74  21 8B 45 08 8B 88 20 03  % ...à└t !ïE.ïê .
0015 EFE0: 00 00 8B 55 FC 89 91 B0  03 00 00 8B 45 08 8B 88  ..ïU"ëæ░ ...ïE.ïê
0015 EFF0: 10 03 00 00 8B 55 FC 89  51 34 8B 45 08 8B 88 20  ....ïU"ë Q4ïE.ïê
0015 F000: 41 39 46 39 62 39 8D 39  92 39 AE 39 D0 39 D5 39  A9F9b9ì9 Æ9«9╨9╒9
0015 F010: F1 39 11 3A 29 3A 3A 3A  3F 3A 5B 3A 7F 3A 91 3A  ±9.:)::: ?:[:⌂:æ:
0015 F020: 9C 3A A1 3A BD 3A E3 3A  F4 3A F9 3A 15 3B 35 3B  £:í:╜:π: ⌠:·:.;5;
0015 F030: 8B 3B 90 3B 9E 3B DD 3B  E2 3B F0 3B 62 3C 6F 3C  ï;É;₧;▌; Γ;≡;b<o<

Set3\test
0015 EFB0: 8B 82 20 03 00 00 88 88  AC 03 00 00 8D 4D FC 51  ïé ...êê ¼...ìM"Q
0015 EFC0: 6A 04 68 B4 AC 1C 8B 8B  55 08 52 E8 F1 FD FF FF  j.h┤¼.ïï U.RΦ±²
0015 EFD0: 25 FF 00 00 00 85 C0 74  21 8B 45 08 8B 88 20 03  % ...à└t !ïE.ïê .
0015 EFE0: 00 00 8B 55 FC 89 91 B0  03 00 00 8B 45 08 8B 88  ..ïU"ëæ░ ...ïE.ïê
0015 EFF0: 10 03 00 00 8B 55 FC 89  51 34 8B 45 08 8B 88 20  ....ïU"ë Q4ïE.ïê
0015 F000: 68 62 69 6E 00 E0 B1 01  00 10 00 00 00 00 00 00  hbin.α▒. ........
0015 F010: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0015 F020: A0 FF FF FF 6E 6B 20 00  30 EE B1 D3 EE C0 C8 01  á   nk . 0ε▒╙ε└╚.
0015 F030: 00 00 00 00 E8 DE B1 01  01 00 00 00 00 00 00 00  ....Φ▐▒. ........

 Arrow keys move   F find      RET next difference   ESC quit   ALT  freeze top
 C ASCII/EBCDIC    E edit file   G goto position       Q quit   CTRL freeze bottom
```

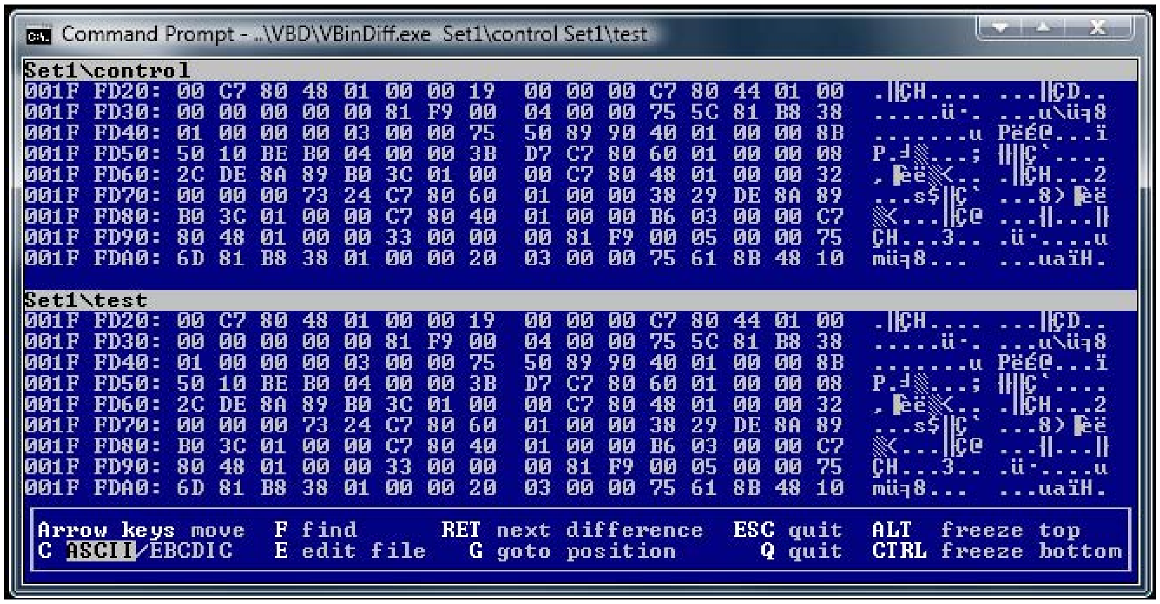
Figure 21. Set 3: Position (0x0015F000) where images start to differ

Bringing the discussion back to the images in set 1, the rest of the images from position 0x0014C000 onwards are patches where data is different – see Figure 22, and patches where data match – see Figure 23. This is expected, as the Operating System would have changed some of the data in RAM – the patches where data is different, but not all of the data in RAM – thereby leaving some areas where data still do match. One of the areas where data do match contains the information that was deliberately inserted into memory. Figure 24 shows the earliest location, 0x005CA012, where either of the "known strings" is found. This is a match, albeit expected, to the first address in the red boxes of Figure 9 and 10. Upon closer inspection of the two red boxes in Figure 9 and 10 and Figure 24, both 010 Editor and VBindiff reveal that the "known strings" are occupying the same addresses in the two images



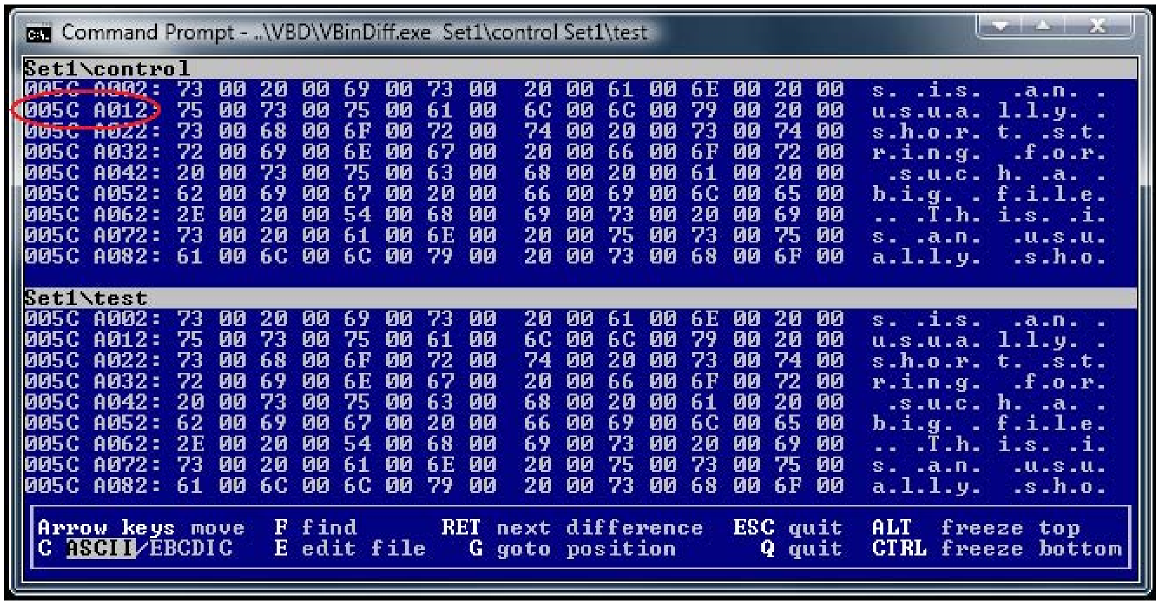
```
Command Prompt - ..\VBD\VBinDiff.exe  Set1\control Set1\test
Set1\control
001F 14C0: 57 FF D6 85 C0 A3 EC 61  AF 6D 0F 84 E7 01 00 00  W ╓à└ú∞a »m.äτ...
001F 14D0: 68 D0 42 AC 6D 57 FF D6  85 C0 A3 F0 61 AF 6D 0F  h╨B¼mW ╓ à└ú≡a»m.
001F 14E0: 84 D2 01 00 00 68 BC 42  AC 6D 57 FF D6 85 C0 A3  ä╥...h╝B ¼mW ╓à└ú
001F 14F0: F4 61 AF 6D 0F 84 BD 01  00 00 68 A8 42 AC 6D 57  ⌠a»m.ä╜. ..h¿B¼mW
001F 1500: FF D6 85 C0 A3 F8 61 AF  6D 0F 84 A8 01 00 00 68   ╓à└ú°a» m.ä¿...h
001F 1510: 8C 42 AC 6D 57 FF D6 85  C0 A3 FC 61 AF 6D 0F 84  îB¼mW ╓à └úⁿa»m.ä
001F 1520: 93 01 00 00 68 7C 42 AC  6D 57 FF D6 85 C0 A3 00  ô...h|B¼ mW ╓à└ú.
001F 1530: 62 AF 6D 0F 84 7E 01 00  00 68 60 42 AC 6D 57 FF  b»m.ä~.. .h`B¼mW
001F 1540: D6 85 C0 A3 04 62 AF 6D  0F 84 69 01 00 00 68 4C  ╓à└ú.b»m .äi...hL

Set1\test
001F 14C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
001F 14D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
001F 14E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
001F 14F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
001F 1500: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
001F 1510: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
001F 1520: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
001F 1530: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
001F 1540: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........

 Arrow keys move   F find       RET next difference   ESC quit   ALT  freeze top
 C ASCII/EBCDIC    E edit file    G goto position       Q quit   CTRL freeze bottom
```

Figure 22. Set 1: An area (0x001F14C0) where data do not match



```
Command Prompt - ..\VBD\VBinDiff.exe  Set1\control Set1\test
Set1\control
001F FD20: 00 C7 80 48 01 00 00 19  00 00 00 C7 80 44 01 00  .╟ÇH.... ...╟ÇD..
001F FD30: 00 00 00 00 00 81 F9 00  04 00 00 75 5C 81 B8 38  .....ü∙. ...u\ü╕8
001F FD40: 01 00 00 00 03 00 00 75  50 89 90 40 01 00 00 8B  .......u PëÉ@...ï
001F FD50: 50 10 BE B0 04 00 00 3B  D7 C7 80 60 01 00 00 08  P.╛░...; ╫╟Ç`....
001F FD60: 2C DE 8A 89 B0 3C 01 00  00 C7 80 48 01 00 00 32  ,▐èë░<.. .╟ÇH...2
001F FD70: 00 00 00 73 24 C7 80 60  01 00 00 38 29 DE 8A 89  ...s$╟Ç` ...8)▐èë
001F FD80: B0 3C 01 00 00 C7 80 40  01 00 00 B6 03 00 00 C7  ░<...╟Ç@ ...╢...╟
001F FD90: 80 48 01 00 00 33 00 00  00 81 F9 00 05 00 00 75  ÇH...3.. .ü∙....u
001F FDA0: 6D 81 B8 38 01 00 00 20  03 00 00 75 61 8B 48 10  mü╕8...  ...uaïH.

Set1\test
001F FD20: 00 C7 80 48 01 00 00 19  00 00 00 C7 80 44 01 00  .╟ÇH.... ...╟ÇD..
001F FD30: 00 00 00 00 00 81 F9 00  04 00 00 75 5C 81 B8 38  .....ü∙. ...u\ü╕8
001F FD40: 01 00 00 00 03 00 00 75  50 89 90 40 01 00 00 8B  .......u PëÉ@...ï
001F FD50: 50 10 BE B0 04 00 00 3B  D7 C7 80 60 01 00 00 08  P.╛░...; ╫╟Ç`....
001F FD60: 2C DE 8A 89 B0 3C 01 00  00 C7 80 48 01 00 00 32  ,▐èë░<.. .╟ÇH...2
001F FD70: 00 00 00 73 24 C7 80 60  01 00 00 38 29 DE 8A 89  ...s$╟Ç` ...8)▐èë
001F FD80: B0 3C 01 00 00 C7 80 40  01 00 00 B6 03 00 00 C7  ░<...╟Ç@ ...╢...╟
001F FD90: 80 48 01 00 00 33 00 00  00 81 F9 00 05 00 00 75  ÇH...3.. .ü∙....u
001F FDA0: 6D 81 B8 38 01 00 00 20  03 00 00 75 61 8B 48 10  mü╕8...  ...uaïH.

 Arrow keys move   F find       RET next difference   ESC quit   ALT  freeze top
 C ASCII/EBCDIC    E edit file    G goto position       Q quit   CTRL freeze bottom
```

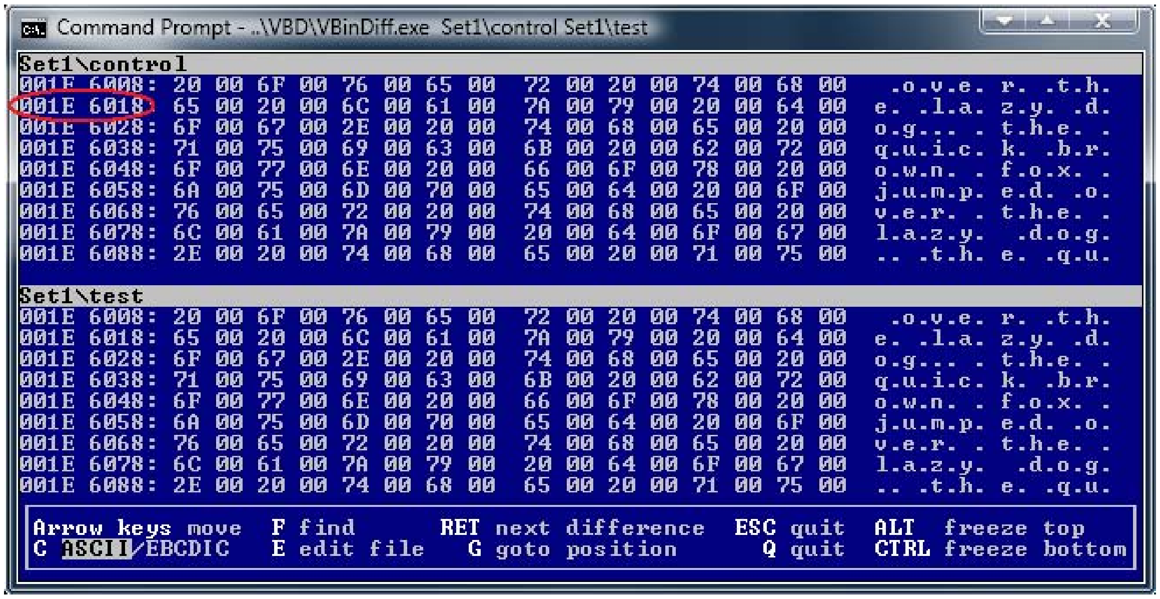
Figure 23. Set 1: An area (0x001FFD20) where the data match



```
Command Prompt - ..\VBD\VBinDiff.exe  Set1\control Set1\test
Set1\control
005C A002: 73 00 20 00 69 00 73 00  20 00 61 00 6E 00 20 00   s. .i.s.  .a.n. .
005C A012: 75 00 73 00 75 00 61 00  6C 00 6C 00 79 00 20 00   u.s.u.a. l.l.y. .
005C A022: 73 00 68 00 6F 00 72 00  74 00 20 00 73 00 74 00   s.h.o.r. t. .s.t.
005C A032: 72 00 69 00 6E 00 67 00  20 00 66 00 6F 00 72 00   r.i.n.g.  .f.o.r.
005C A042: 20 00 73 00 75 00 63 00  68 00 20 00 61 00 20 00    .s.u.c. h. .a. .
005C A052: 62 00 69 00 67 00 20 00  66 00 69 00 6C 00 65 00   b.i.g. . f.i.l.e.
005C A062: 2E 00 20 00 54 00 68 00  69 00 73 00 20 00 69 00   .. .T.h. i.s. .i.
005C A072: 73 00 20 00 61 00 6E 00  20 00 75 00 73 00 75 00   s. .a.n.  .u.s.u.
005C A082: 61 00 6C 00 6C 00 79 00  20 00 73 00 68 00 6F 00   a.l.l.y.  .s.h.o.

Set1\test
005C A002: 73 00 20 00 69 00 73 00  20 00 61 00 6E 00 20 00   s. .i.s.  .a.n. .
005C A012: 75 00 73 00 75 00 61 00  6C 00 6C 00 79 00 20 00   u.s.u.a. l.l.y. .
005C A022: 73 00 68 00 6F 00 72 00  74 00 20 00 73 00 74 00   s.h.o.r. t. .s.t.
005C A032: 72 00 69 00 6E 00 67 00  20 00 66 00 6F 00 72 00   r.i.n.g.  .f.o.r.
005C A042: 20 00 73 00 75 00 63 00  68 00 20 00 61 00 20 00    .s.u.c. h. .a. .
005C A052: 62 00 69 00 67 00 20 00  66 00 69 00 6C 00 65 00   b.i.g. . f.i.l.e.
005C A062: 2E 00 20 00 54 00 68 00  69 00 73 00 20 00 69 00   .. .T.h. i.s. .i.
005C A072: 73 00 20 00 61 00 6E 00  20 00 75 00 73 00 75 00   s. .a.n.  .u.s.u.
005C A082: 61 00 6C 00 6C 00 79 00  20 00 73 00 68 00 6F 00   a.l.l.y.  .s.h.o.

 Arrow keys move  F find      RET next difference  ESC quit  ALT  freeze top
 C ASCII/EBCDIC   E edit file   G goto position       Q quit  CTRL freeze bottom
```

Figure 24. Set 1: The first instance of the signature "usually"

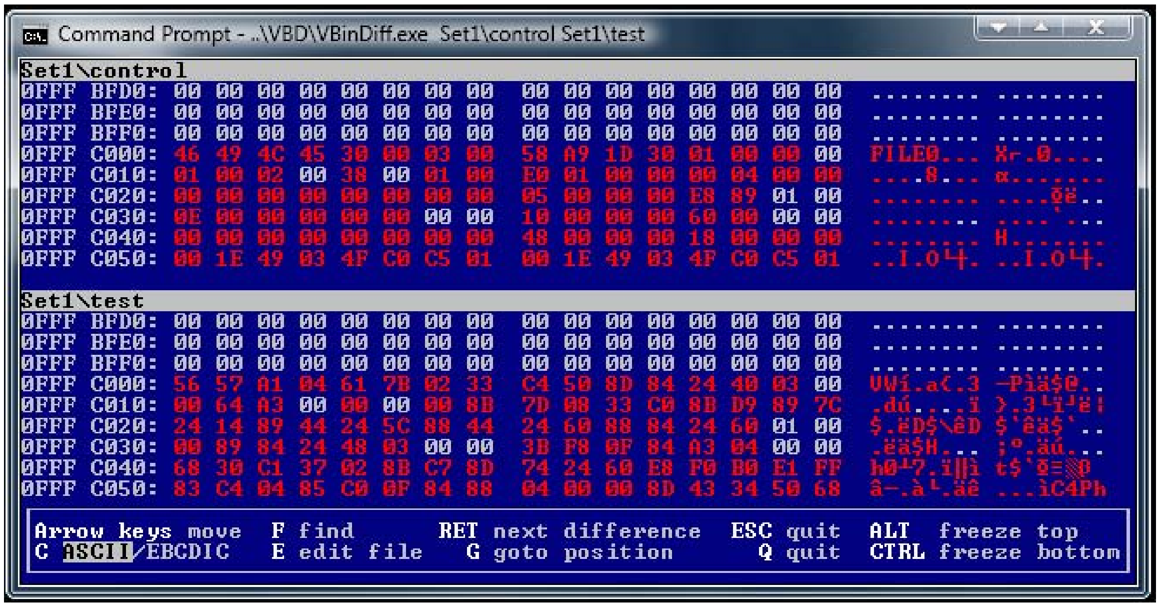
Figure 25 show the VBinDiff output of the earliest instance of the signature "*quick brown*". Not only does the segment of memory match in both images, this is consistent with the results boxed in red of Figure 11 and Figure 12.



```
Command Prompt - ..\VBD\VBinDiff.exe  Set1\control Set1\test
Set1\control
001E 6008: 20 00 6F 00 76 00 65 00  72 00 20 00 74 00 68 00    .o.v.e. r. .t.h.
001E 6018: 65 00 20 00 6C 00 61 00  7A 00 79 00 20 00 64 00   e. .l.a. z.y. .d.
001E 6028: 6F 00 67 00 2E 00 20 00  74 00 68 00 65 00 20 00   o.g... . t.h.e. .
001E 6038: 71 00 75 00 69 00 63 00  6B 00 20 00 62 00 72 00   q.u.i.c. k. .b.r.
001E 6048: 6F 00 77 00 6E 00 20 00  66 00 6F 00 78 00 20 00   o.w.n. . f.o.x. .
001E 6058: 6A 00 75 00 6D 00 70 00  65 00 64 00 20 00 6F 00   j.u.m.p. e.d. .o.
001E 6068: 76 00 65 00 72 00 20 00  74 00 68 00 65 00 20 00   v.e.r. . t.h.e. .
001E 6078: 6C 00 61 00 7A 00 79 00  20 00 64 00 6F 00 67 00   l.a.z.y.  .d.o.g.
001E 6088: 2E 00 20 00 74 00 68 00  65 00 20 00 71 00 75 00   .. .t.h. e. .q.u.

Set1\test
001E 6008: 20 00 6F 00 76 00 65 00  72 00 20 00 74 00 68 00    .o.v.e. r. .t.h.
001E 6018: 65 00 20 00 6C 00 61 00  7A 00 79 00 20 00 64 00   e. .l.a. z.y. .d.
001E 6028: 6F 00 67 00 2E 00 20 00  74 00 68 00 65 00 20 00   o.g... . t.h.e. .
001E 6038: 71 00 75 00 69 00 63 00  6B 00 20 00 62 00 72 00   q.u.i.c. k. .b.r.
001E 6048: 6F 00 77 00 6E 00 20 00  66 00 6F 00 78 00 20 00   o.w.n. . f.o.x. .
001E 6058: 6A 00 75 00 6D 00 70 00  65 00 64 00 20 00 6F 00   j.u.m.p. e.d. .o.
001E 6068: 76 00 65 00 72 00 20 00  74 00 68 00 65 00 20 00   v.e.r. . t.h.e. .
001E 6078: 6C 00 61 00 7A 00 79 00  20 00 64 00 6F 00 67 00   l.a.z.y.  .d.o.g.
001E 6088: 2E 00 20 00 74 00 68 00  65 00 20 00 71 00 75 00   .. .t.h. e. .q.u.

 Arrow keys move  F find      RET next difference  ESC quit  ALT  freeze top
 C ASCII/EBCDIC   E edit file   G goto position       Q quit  CTRL freeze bottom
```

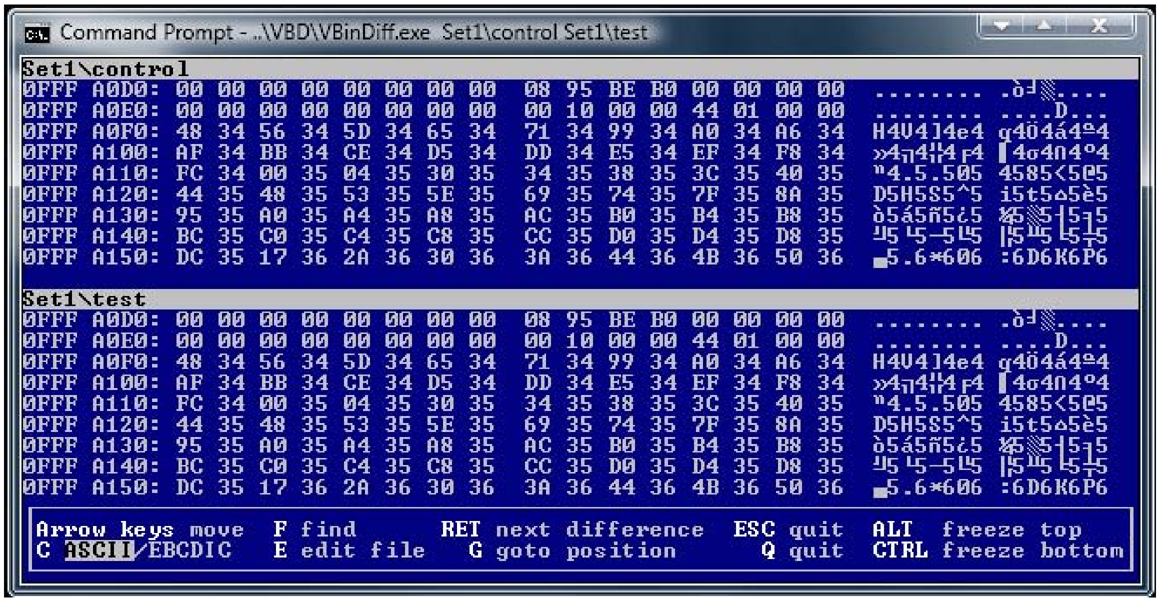
Figure 25. Set 1: The first instance of the signature "quick brown"

Figure 26 and Figure 27 shows the pattern of matching and unmatching segments of memory continue further in the images of set 1. This pattern continues to the end of the images.



```
Command Prompt - ..\VBD\VBinDiff.exe  Set1\control Set1\test
Set1\control
0FFF BFD0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0FFF BFE0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0FFF BFF0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0FFF C000: 46 49 4C 45 30 00 03 00  58 A9 1D 30 01 00 00 00  FILE0... Xr.0....
0FFF C010: 01 00 02 00 38 00 01 00  E0 01 00 00 00 04 00 00  ....8... α.......
0FFF C020: 00 00 00 00 00 00 00 00  05 00 00 00 E8 89 01 00  ........ ....Şë..
0FFF C030: 0E 00 00 00 00 00 00 00  10 00 00 00 60 00 00 00  ........ ....`...
0FFF C040: 00 00 00 00 00 00 00 00  48 00 00 00 18 00 00 00  ........ H.......
0FFF C050: 00 1E 49 03 4F C0 C5 01  00 1E 49 03 4F C0 C5 01  ..I.O└┼. ..I.O└┼.

Set1\test
0FFF BFD0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0FFF BFE0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0FFF BFF0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
0FFF C000: 56 57 A1 04 61 7B 02 33  C4 50 8D 84 24 40 03 00  VWí.a{.3 ─Pìä$@..
0FFF C010: 00 64 A3 00 00 00 00 8B  7D 08 33 C0 8B D9 89 7C  .dú....ï }.3└ï┘ë|
0FFF C020: 24 14 89 44 24 5C 88 44  24 60 88 84 24 60 01 00  $.ëD$\êD $`êä$`..
0FFF C030: 00 89 84 24 48 03 00 00  3B F8 0F 84 A3 04 00 00  .ëä$H... ;°.äú...
0FFF C040: 68 30 C1 37 02 8B C7 8D  74 24 60 E8 F0 B0 E1 FF  h0┴7.ï╟ì t$`Φ≡░ß 
0FFF C050: 83 C4 04 85 C0 0F 84 88  04 00 00 8D 43 34 50 68  â─.à└.äê ...ìC4Ph

 Arrow keys move   F find       RET next difference   ESC quit   ALT  freeze top
 C ASCII/EBCDIC    E edit file    G goto position       Q quit   CTRL freeze bottom
```

Figure 26. Set 1: Another area (0x0FFFC000) where data do not match



```
Command Prompt - ..\VBD\VBinDiff.exe  Set1\control Set1\test
Set1\control
0FFF A0D0: 00 00 00 00 00 00 00 00  08 95 BE B0 00 00 00 00  ........ .ò╛░....
0FFF A0E0: 00 00 00 00 00 00 00 00  00 10 00 00 44 01 00 00  ........ ....D...
0FFF A0F0: 48 34 56 34 5D 34 65 34  71 34 99 34 A0 34 A6 34  H4V4]4e4 q4Ö4á4ª4
0FFF A100: AF 34 BB 34 CE 34 D5 34  DD 34 E5 34 EF 34 F8 34  »4╗4╬4╒4 ▌4σ4∩4°4
0FFF A110: FC 34 00 35 04 35 30 35  34 35 38 35 3C 35 40 35  ⁿ4.5.505 4585<5@5
0FFF A120: 44 35 48 35 53 35 5E 35  69 35 74 35 7F 35 8A 35  D5H5S5^5 i5t5⌂5è5
0FFF A130: 95 35 A0 35 A4 35 A8 35  AC 35 B0 35 B4 35 B8 35  ò5á5ñ5¿5 ¼5░5┤5╕5
0FFF A140: BC 35 C0 35 C4 35 C8 35  CC 35 D0 35 D4 35 D8 35  ╝5└5─5╚5 ╠5╨5╘5╪5
0FFF A150: DC 35 17 36 2A 36 30 36  3A 36 44 36 4B 36 50 36  ▄5.6*606 :6D6K6P6

Set1\test
0FFF A0D0: 00 00 00 00 00 00 00 00  08 95 BE B0 00 00 00 00  ........ .ò╛░....
0FFF A0E0: 00 00 00 00 00 00 00 00  00 10 00 00 44 01 00 00  ........ ....D...
0FFF A0F0: 48 34 56 34 5D 34 65 34  71 34 99 34 A0 34 A6 34  H4V4]4e4 q4Ö4á4ª4
0FFF A100: AF 34 BB 34 CE 34 D5 34  DD 34 E5 34 EF 34 F8 34  »4╗4╬4╒4 ▌4σ4∩4°4
0FFF A110: FC 34 00 35 04 35 30 35  34 35 38 35 3C 35 40 35  ⁿ4.5.505 4585<5@5
0FFF A120: 44 35 48 35 53 35 5E 35  69 35 74 35 7F 35 8A 35  D5H5S5^5 i5t5⌂5è5
0FFF A130: 95 35 A0 35 A4 35 A8 35  AC 35 B0 35 B4 35 B8 35  ò5á5ñ5¿5 ¼5░5┤5╕5
0FFF A140: BC 35 C0 35 C4 35 C8 35  CC 35 D0 35 D4 35 D8 35  ╝5└5─5╚5 ╠5╨5╘5╪5
0FFF A150: DC 35 17 36 2A 36 30 36  3A 36 44 36 4B 36 50 36  ▄5.6*606 :6D6K6P6

 Arrow keys move   F find       RET next difference   ESC quit   ALT  freeze top
 C ASCII/EBCDIC    E edit file    G goto position       Q quit   CTRL freeze bottom
```

Figure 27. Set 1: Another area (0x0FFFA0D0) where the data match

VII. FUTURE WORK

The goal of this research is an attempt to show the possibility of creating an image of volatile RAM using only user mode privileges, and the help of a custom preinstalled windows driver. It is far from being a completed and finished product and below are some possibilities of how this is research can be further polished and perpetuated.

A. STABILITY OF THE MEMREADER DRIVER

As mentioned in previous chapters, the MemReader driver should not be considered a production quality product. The source of crashes needs to be determined. This will require “live” debugging to be done and likely a detailed understanding of memory management of the Windows kernel. Once the stability issue is addresses, the driver can used as a commercially viable package.

B. POWER MANAGEMENT EVENT CALLBACKS

There are two power callback events currently registered in the MemReader driver, the DeviceSelfManagedIoStart and the DeviceSelfManagedIoSuspend events. These are shell functions without implementation. Implementing these functions would complete the driver and make the environment where it operates be more stable, especially when there is a power transition event when it is operating.

C. TESTING ON DIFFERENT FLAVORS OF WINDOWS VISTA

There are four flavors to the Windows Vista Operating System, the Home, Home Premium, Business and Ultimate and there is a 64-bit version that takes advantage of the 64-bit processors. Since the development was done on a System running Windows Vista Business, testing was not done on the other flavors and versions of Windows Vista. The driver was also not tested on any other hardware architecture except for the x86 architecture (the development environment).

D. WRITE CAPABILITY (SIMILAR TO DD)

Since the physical memory segment is fully accessible by the Memreader device driver, its functionality can be expanded to include write capabilities much like what the UNIX command dd can do. Currently the physical memory section is opened and mapped with a READ_ONLY attribute, this needs to be changed to include the possibility of writing to it. The write command can be implemented in many ways. This can include registering another callback for I/O control events or simply by modifying the memaddr data structure to include addition parameters needed to initiate the write operation.

The read and write capability can be further extended to files and even blocks and sectors of the hard disk, much like dd. Adding a write capability can result in obvious negative effects for forensic purposes.

VIII. CONCLUSIONS AND RECOMMENDATIONS

The main objective of this research is to show that, with a suitable device driver, it is possible for a user mode process to create an image of RAM. In order to verify that the MemReader driver and application pair is functioning correctly, it must be shown that the image created using the MemReader driver and application is correct.

It is impossible to determine the exact content of RAM when the MemReader driver and application was creating the image. In fact, the very operation of the MemReader driver and application will alter the content of memory. Therefore, it is only possible to verify to a certain degree that the information captured is correct and accurate. For the driver to work seamlessly, an in-depth knowledge of Windows Vista is needed. The proprietary nature of Windows operating system makes this a tricky task as many hypotheses are formed and testing is needed. It is however not unfathomable that such a driver can be written.

The results clearly shows that the image created by the MemReader driver and the user mode MemReader application has capture the consistent part of RAM, which mainly consist of machine and system related information. The image created by MemReader also captured the injected known strings at position where they are expected. There is strong evidence that the driver and application pair in able to create an image of RAM.

The results achieved are satisfactory and they clearly show that it is possible to create an image of RAM using a user-mode application with the help of a suitable, pre-installed device driver. The objectives of the research are therefore met.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A: INSTRUCTIONS TO SETTING UP A SIGNING ENVIRONMENT

Test Signing [35]

1. Prepare a computer for signing
    a. Install WDK
2. Create a test certificate and adding it to the Trusted Root certificate store and the Trusted Publishers certificate store
    a. Makecert -r -pe -ss *SubjectCertStoreName* -n “CN=*CertName*” *OutputFile*.cer
    b. certmgr.exe -add OutputFile.cer -s -r localMachine root
    c. certmgr.exe -add OutputFile.cer -s -r localMachine trustedpublisher
3. Create a catalog file using Inf2Cat[9]
    a. Create a driver package directory that contains all of the files in your driver package.
    b. Create an INF file in the driver package directory and edit it for Windows Vista. Change the build date and the version number to 6. For example: DriverVer=12/04/2008, 6.0.1.0
    c. Inf2cat.exe /driver:C:\WinDDK\6000\src\kmdf\memreader\ /os:Vista_x86

Test-sign the catalog file using SignTool

SignTool sign /v /ac *CrossCertificateFile* /s *SPCCertificateStore* /n *SPCSubjectName* /t http://timestamp.verisign.com/scripts/timestamp.dll Good.cat

[9] Registration is required at the Winqual website before the Inf2Cat application is available for download.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B: INSTRUCTIONS TO SETUP A KERNEL DEVELOPMENT ENVIRONMENT

1. Download and install the Windows Driver Kit (WDK)
    a. WDK is available for download the Microsoft Connect website, http://connect.microsoft.com/. Registration is required before the WDK is available for download.
    b. Upon logging into Microsoft connect, browse to the connections directory and join the Windows Driver Kit (WDK), Windows Logo Kit (WLK) and Windows Driver Framework (WDF)[10].
    c. The WDK installation package can be downloaded from the Windows Driver Kit (WDK), Windows Logo Kit (WLK) and Windows Driver Framework (WDF) connection page.
    d. Run the downloaded executable and install WDK.
2. Download and install DebugView
    a. DebugView is freely available for download from the Microsoft Technet, http://technet.microsoft.com/en-us/sysinternals/bb896647.aspx.
    b. Run the downloaded executable to install DebugView.
    c. DebugView requires elevated privileges to function. Right click on the application shortcut and select the 'Run as administrator' option to run DebugView.

[10] If this connection is not available for joining, a request can be made to the administrator through the connection directory help.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C: INSTRUCTIONS FOR INSTALLING AND UNINSTALLING WINDOWS DRIVER

INSTALLATION (x86 Free Build Environment)

1. Build the driver files
2. Copy the WDF Co-installer files (WdfCoInstaller<*version#*>.dll) from the <*WDKROOT*>\redist\wdf\x86 directory to directory where the build output files (INF, .sys, .pdb, .mac and .obj files) are found.
3. Open Control Panel > Add Hardware menu (Administrator Privileged needed)
4. At the Add Hardware wizard select “Next >”
5. Select the “Install the hardware that I manually select from a list (Advance)” radio button then hit “Next >”
6. Select “Show All Devices” and hit “Next >”
7. Select “Have Disk…”
8. In the “Install From Disk” menu select “Browse…”
9. In the “Locate File” menu, navigate to the directory where the outputs files are located and select the installation (INF) file then select “Open”
10. Select “OK” and hit “Next >”
11. Hit “Next >” again
12. If a “Windows Security” alert pops up, Select the option “Install this driver software anyway”
13. Select “Finish” to complete the installation

UNINSTALLATION

1. Open Control Panel > System > Device Manager menu (Administrator Privileged needed)

2. Locate and select the device node in the “Device Manager” Menu
3. Right click on the selected device node and select “Uninstall”
4. In the “Confirm Device Uninstall” menu check the “Delete the driver software for this device” checkbox and hit “OK”

APPENDIX D: INSTRUCTIONS FOR CREATING A RAM IMAGE

1. Start the system and log into the user account
2. Open a text file with a repeated (file size ~150MB) known string (e.g. a quick brown fox jumped over the lazy dog).

Creating a RAM image using ManTech's DD

3. Open a Command Prompt with administrator privileges
    a. Click on the start Windows Key and select Programs > Accessories > Command Prompt
    b. Right click of the highlighted Command Prompt and select "Run as administrator"
4. Navigate to the directory where the ManTech's DD tool is installed
    c. cd C:\MDD
5. Create an image file of the RAM
    d. mdd_v1.3.exe –o *control.bin*

Creating a RAM image using MemReader

3. Open a Command Prompt
    a. Click on the start Windows Key and select Programs > Accessories > Command Prompt and click on/ select it
4. Navigate to the directory where the MemReader application is installed
    a. cd C:\MemReader
5. Create an image file of the RAM
    a. MemReader.exe *test.bin* <*offset*>

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E: FULL SOURCE CODE AND INSTALLATION FILES

The source code of MemReader is licensed under the BSD license.

public.h

```
/*
* Copyright (c) 2008, Vincent Cheong
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
are met:
*     * Redistributions of source code must retain the above copyright
*       notice, this list of conditions and the following disclaimer.
*     * Redistributions in binary form must reproduce the above
copyright
*       notice, this list of conditions and the following disclaimer in
the
*       documentation and/or other materials provided with the
distribution.
*     * Neither the name of the Naval Postgraduate School nor the
*       names of its contributors may be used to endorse or promote
products
*       derived from this software without specific prior written
permission.
*
* THIS SOFTWARE IS PROVIDED BY VINCENT CHEONG ''AS IS'' AND ANY
* EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED
* WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE
* DISCLAIMED. IN NO EVENT SHALL <copyright holder> BE LIABLE FOR ANY
* DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES
* (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES;
* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND
* ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
TORT
* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS
* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

#define WHILE(a) \
while(__pragma(warning(disable:4127)) a
__pragma(warning(disable:4127)))

#define TX_BLOCK_SIZE 4096
```

```
// Define an Interface Guid so that app can find the device and talk to
it.
DEFINE_GUID (GUID_DEVINTERFACE_ECHO,
    0xcdc35b6e, 0xbe4, 0x4936, 0xbf, 0x5f, 0x55, 0x37, 0x38, 0xa, 0x7c,
0x1a);
// {CDC35B6E-0BE4-4936-BF5F-5537380A7C1A}
```

driver.h

```
/*
* Copyright (c) 2008, Vincent Cheong
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
are met:
*     * Redistributions of source code must retain the above copyright
*       notice, this list of conditions and the following disclaimer.
*     * Redistributions in binary form must reproduce the above
copyright
*       notice, this list of conditions and the following disclaimer in
the
*       documentation and/or other materials provided with the
distribution.
*     * Neither the name of the Naval Postgraduate School nor the
*       names of its contributors may be used to endorse or promote
products
*       derived from this software without specific prior written
permission.
*
* THIS SOFTWARE IS PROVIDED BY VINCENT CHEONG ''AS IS'' AND ANY
* EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED
* WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE
* DISCLAIMED. IN NO EVENT SHALL <copyright holder> BE LIABLE FOR ANY
* DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES
* (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES;
* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND
* ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
TORT
* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS
* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

#define INITGUID

#include <ntddk.h>
#include <wdf.h>

#include "device.h"
#include "queue.h"

// WDFDRIVER Events
DRIVER_INITIALIZE DriverEntry;

NTSTATUS
MemReaderEvtDeviceAdd(
    IN WDFDRIVER Driver,
    IN PWDFDEVICE_INIT DeviceInit
    );

NTSTATUS
MemReaderPrintDriverVersion(
    );
```

driver.c

```
/*
* Copyright (c) 2008, Vincent Cheong
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
are met:
*     * Redistributions of source code must retain the above copyright
*       notice, this list of conditions and the following disclaimer.
*     * Redistributions in binary form must reproduce the above
copyright
*       notice, this list of conditions and the following disclaimer in
the
*       documentation and/or other materials provided with the
distribution.
*     * Neither the name of the Naval Postgraduate School nor the
*       names of its contributors may be used to endorse or promote
products
*       derived from this software without specific prior written
permission.
*
* THIS SOFTWARE IS PROVIDED BY VINCENT CHEONG ''AS IS'' AND ANY
* EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED
* WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE
* DISCLAIMED. IN NO EVENT SHALL <copyright holder> BE LIABLE FOR ANY
* DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES
* (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES;
* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND
* ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
TORT
* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS
* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

#include "driver.h"

#ifdef ALLOC_PRAGMA
#pragma alloc_text (INIT, DriverEntry)
#pragma alloc_text (INIT, MemReaderPrintDriverVersion)
#pragma alloc_text (PAGE, MemReaderEvtDeviceAdd)
#endif


NTSTATUS
DriverEntry(
    IN PDRIVER_OBJECT  DriverObject,
    IN PUNICODE_STRING RegistryPath
    )
{
    NTSTATUS                  status;
    WDF_DRIVER_CONFIG   config;

    WDF_DRIVER_CONFIG_INIT(&config,
                        MemReaderEvtDeviceAdd
                        );

    status = WdfDriverCreate(DriverObject,
                            RegistryPath,
                            WDF_NO_OBJECT_ATTRIBUTES,
                            &config,
                            WDF_NO_HANDLE);
    if (!NT_SUCCESS(status)) {
            DbgPrint("MEM: Error: WdfDriverCreate failed 0x%x\n",
status);
        return status;
    }

#if DBG
    EchoPrintDriverVersion();
#endif

    return status;
}

NTSTATUS
MemReaderEvtDeviceAdd(
    IN WDFDRIVER       Driver,
    IN PWDFDEVICE_INIT DeviceInit
    )
{
    NTSTATUS status;

    UNREFERENCED_PARAMETER(Driver);

    PAGED_CODE();

      DbgPrint("MEM: Enter  EchoEvtDeviceAdd\n");

    status = MemReaderDeviceCreate(DeviceInit);

    return status;
}

NTSTATUS
MemReaderPrintDriverVersion(
    )
{
    NTSTATUS                                        status;
    WDFSTRING                                       string;
    UNICODE_STRING                                  us;
    WDF_DRIVER_VERSION_AVAILABLE_PARAMS   ver;
    // Retreive version string and print that in the debugger.
    status = WdfStringCreate(NULL, WDF_NO_OBJECT_ATTRIBUTES, &string);
    if (!NT_SUCCESS(status)) {
        DbgPrint("Error: WdfStringCreate failed 0x%x\n", status);
        return status;
    }

    status = WdfDriverRetrieveVersionString(WdfGetDriver(), string);
    if (!NT_SUCCESS(status)) {
        //
        // No need to worry about delete the string object because
        // by default it's parented to the driver and it will be
        // deleted when the driverobject is deleted when the
DriverEntry
        // returns a failure status.
        //
        DbgPrint("Error: WdfDriverRetrieveVersionString failed 0x%x\n",
status);
        return status;
    }

    WdfStringGetUnicodeString(string, &us);
      DbgPrint("MEM: MemReader %wZ, built on %s %s\n", &us,
            __DATE__, __TIME__);

    WdfObjectDelete(string);
    string = NULL; // To avoid referencing a deleted object.

    // Find out to which version of framework this driver is bound to.
    WDF_DRIVER_VERSION_AVAILABLE_PARAMS_INIT(&ver, 1, 0);
    if (WdfDriverIsVersionAvailable(WdfGetDriver(), &ver) == TRUE) {
            DbgPrint("MEM: Yes, framework version is 1.0\n");
    }else {
            DbgPrint("MEM: No, framework verison is not 1.0\n");
    }

    return STATUS_SUCCESS;
}
```

device.h

```
/*
* Copyright (c) 2008, Vincent Cheong
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
are met:
*     * Redistributions of source code must retain the above copyright
*       notice, this list of conditions and the following disclaimer.
*     * Redistributions in binary form must reproduce the above
copyright
*       notice, this list of conditions and the following disclaimer in
the
*       documentation and/or other materials provided with the
distribution.
*     * Neither the name of the Naval Postgraduate School nor the
*       names of its contributors may be used to endorse or promote
products
*       derived from this software without specific prior written
permission.
*
* THIS SOFTWARE IS PROVIDED BY VINCENT CHEONG ''AS IS'' AND ANY
* EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED
* WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE
* DISCLAIMED. IN NO EVENT SHALL <copyright holder> BE LIABLE FOR ANY
* DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES
* (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES;
* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND
* ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
TORT
* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS
* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

#include "..\exe\public.h"

// The device context performs the same job as
// a WDM device extension in the driver frameworks
typedef struct _DEVICE_CONTEXT
{
    ULONG PrivateDeviceData;  // just a placeholder

} DEVICE_CONTEXT, *PDEVICE_CONTEXT;

// This macro will generate an inline function called
WdfObjectGet_DEVICE_CONTEXT
// which will be used to get a pointer to the device context memory
// in a type safe manner.
WDF_DECLARE_CONTEXT_TYPE(DEVICE_CONTEXT)

// Function to initialize the device and its callbacks
NTSTATUS
MemReaderDeviceCreate(
    PWDFDEVICE_INIT DeviceInit
    );

// Device events
VOID
MemReaderEvtDeviceContextCleanup(
    IN WDFDEVICE hDevice
    );

NTSTATUS
MemReaderEvtDeviceSelfManagedIoStart(
    IN  WDFDEVICE Device
    );

NTSTATUS
MemReaderEvtDeviceSelfManagedIoSuspend(
    IN  WDFDEVICE Device
    );
```

device.c

```
/*
* Copyright (c) 2008, Vincent Cheong
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
are met:
*     * Redistributions of source code must retain the above copyright
*       notice, this list of conditions and the following disclaimer.
*     * Redistributions in binary form must reproduce the above
copyright
*       notice, this list of conditions and the following disclaimer in
the
*       documentation and/or other materials provided with the
distribution.
*     * Neither the name of the Naval Postgraduate School nor the
*       names of its contributors may be used to endorse or promote
products
*       derived from this software without specific prior written
permission.
*
* THIS SOFTWARE IS PROVIDED BY VINCENT CHEONG ''AS IS'' AND ANY
* EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED
* WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE
* DISCLAIMED. IN NO EVENT SHALL <copyright holder> BE LIABLE FOR ANY
* DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES
* (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES;
* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND
* ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
TORT
* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS
* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

#include "driver.h"

#ifdef ALLOC_PRAGMA
#pragma alloc_text (PAGE, MemReaderDeviceCreate)
#pragma alloc_text (PAGE, MemReaderEvtDeviceSelfManagedIoStart)
#pragma alloc_text (PAGE, MemReaderEvtDeviceSelfManagedIoSuspend)
#endif


NTSTATUS
MemReaderDeviceCreate(
    PWDFDEVICE_INIT DeviceInit
    )
```

```
{
    WDF_OBJECT_ATTRIBUTES   deviceAttributes;
    PDEVICE_CONTEXT deviceContext;
    WDF_PNPPOWER_EVENT_CALLBACKS    pnpPowerCallbacks;
    WDFDEVICE device;
    NTSTATUS status;

    PAGED_CODE();

    WDF_PNPPOWER_EVENT_CALLBACKS_INIT(&pnpPowerCallbacks);

    // Register pnp/power callbacks so that we can start and stop the
timer as the device
    // gets started and stopped.
    pnpPowerCallbacks.EvtDeviceSelfManagedIoInit    =
MemReaderEvtDeviceSelfManagedIoStart;
    pnpPowerCallbacks.EvtDeviceSelfManagedIoSuspend =
MemReaderEvtDeviceSelfManagedIoSuspend;
    pnpPowerCallbacks.EvtDeviceSelfManagedIoRestart =
MemReaderEvtDeviceSelfManagedIoStart;

    // Register the PnP and power callbacks.
    WdfDeviceInitSetPnpPowerEventCallbacks(DeviceInit,
&pnpPowerCallbacks);

    WDF_OBJECT_ATTRIBUTES_INIT_CONTEXT_TYPE(&deviceAttributes,
DEVICE_CONTEXT);

      // Set WDFDEVICE synchronization scope. By opting for device
level
    // synchronization scope, all the queue and timer callbacks are
    // synchronized with the device-level spinlock.
    deviceAttributes.SynchronizationScope =
WdfSynchronizationScopeDevice;

    status = WdfDeviceCreate(&DeviceInit, &deviceAttributes, &device);

    if (NT_SUCCESS(status)) {
        // Get the device context and initialize it.
WdfObjectGet_DEVICE_CONTEXT is an
        // inline function generated by WDF_DECLARE_CONTEXT_TYPE macro
in the
        // device.h header file. This function will do the type
checking and return
        // the device context. If you pass a wrong object  handle
        // it will return NULL and assert if run under framework
verifier mode.
        deviceContext = WdfObjectGet_DEVICE_CONTEXT(device);
        deviceContext->PrivateDeviceData = 0;

        // Create a device interface so that application can find and
talk
        // to us.
        status = WdfDeviceCreateDeviceInterface(
            device,
            &GUID_DEVINTERFACE_ECHO,
            NULL // ReferenceString
            );

        if (NT_SUCCESS(status)) {
            // Initialize the I/O Package and any Queues
            status = MemReaderQueueInitialize(device);
        }
    }

    return status;
}


NTSTATUS
MemReaderEvtDeviceSelfManagedIoStart(
    IN  WDFDEVICE Device
    )
{
      DbgPrint("MEM: MemReaderEvtDeviceSelfManagedIoInit\n");

    PAGED_CODE();

    WdfIoQueueStart(WdfDeviceGetDefaultQueue(Device));

      DbgPrint( "MEM: MemReaderEvtDeviceSelfManagedIoInit\n");

    return STATUS_SUCCESS;
}

NTSTATUS
MemReaderEvtDeviceSelfManagedIoSuspend(
    IN  WDFDEVICE Device
    )
{
    PAGED_CODE();

      DbgPrint("MEM: MemReaderEvtDeviceSelfManagedIoSuspend\n");

    WdfIoQueueStopSynchronously(WdfDeviceGetDefaultQueue(Device));

      DbgPrint( "MEM: MemReaderEvtDeviceSelfManagedIoSuspend\n");

    return STATUS_SUCCESS;
}
```

queue.h

```
/*
* Copyright (c) 2008, Vincent Cheong
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
are met:
*     * Redistributions of source code must retain the above copyright
*       notice, this list of conditions and the following disclaimer.
*     * Redistributions in binary form must reproduce the above
copyright
*       notice, this list of conditions and the following disclaimer in
the
*       documentation and/or other materials provided with the
distribution.
*     * Neither the name of the Naval Postgraduate School nor the
*       names of its contributors may be used to endorse or promote
products
*       derived from this software without specific prior written
permission.
*
* THIS SOFTWARE IS PROVIDED BY VINCENT CHEONG ''AS IS'' AND ANY
* EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED
* WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE
* DISCLAIMED. IN NO EVENT SHALL <copyright holder> BE LIABLE FOR ANY
* DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES
* (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES;
* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND
* ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
TORT
* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS
* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

// Memory Address data structure that is also defined in memreaderapp
typedef struct _memaddr{
    ULONGLONG offset;
    ULONGLONG len;
} memaddr;

// This is the context that can be placed per queue
// and would contain per queue information.
typedef struct _QUEUE_CONTEXT {

      HANDLE            hPhysicalMemory ;
      PVOID       pVirtualAddr      ;
      memaddr           ma ;
    WDFREQUEST  CurrentRequest;

} QUEUE_CONTEXT, *PQUEUE_CONTEXT;

WDF_DECLARE_CONTEXT_TYPE_WITH_NAME(QUEUE_CONTEXT, QueueGetContext)

NTSTATUS
MemReaderQueueInitialize(
    WDFDEVICE hDevice
    );

VOID
MemReaderEvtIoQueueContextDestroy(
    WDFOBJECT Object
);

// Callback Events from the IoQueue object
VOID
MemReaderEvtRequestCancel(
    IN WDFREQUEST Request
    );

VOID
MemReaderEvtIoRead(
    IN WDFQUEUE   Queue,
    IN WDFREQUEST Request,
    IN size_t      Length
    );

VOID
MemReaderEvtIoWrite(
    IN WDFQUEUE   Queue,
    IN WDFREQUEST Request,
    IN size_t      Length
    );
```

queue.c

```
/*
* Copyright (c) 2008, Vincent Cheong
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
are met:
*     * Redistributions of source code must retain the above copyright
*       notice, this list of conditions and the following disclaimer.
*     * Redistributions in binary form must reproduce the above
copyright
*       notice, this list of conditions and the following disclaimer in
the
*       documentation and/or other materials provided with the
distribution.
*     * Neither the name of the Naval Postgraduate School nor the
*       names of its contributors may be used to endorse or promote
products
*       derived from this software without specific prior written
permission.
*
* THIS SOFTWARE IS PROVIDED BY VINCENT CHEONG ''AS IS'' AND ANY
* EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED
* WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE
* DISCLAIMED. IN NO EVENT SHALL <copyright holder> BE LIABLE FOR ANY
* DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES
* (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES;
* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND
* ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
TORT
* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS
* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

#include "driver.h"

#ifdef ALLOC_PRAGMA
#pragma alloc_text (PAGE, MemReaderQueueInitialize)
#endif

NTSTATUS
MemReaderQueueInitialize(
    WDFDEVICE Device
    )
{
    WDFQUEUE                        queue;
    NTSTATUS                        status;
```

```
      UNICODE_STRING                usPhysicalMemory;
    PQUEUE_CONTEXT                  queueContext;
      OBJECT_ATTRIBUTES       oa;
    WDF_IO_QUEUE_CONFIG       queueConfig;
    WDF_OBJECT_ATTRIBUTES     queueAttributes;

    PAGED_CODE();
      DbgPrint("MEM: Queue Initialize X!\n");

    // Configure a default queue so that requests that are not
    // configure-fowarded using WdfDeviceConfigureRequestDispatching to
goto
    // other queues get dispatched here.
    WDF_IO_QUEUE_CONFIG_INIT_DEFAULT_QUEUE(
         &queueConfig,
        WdfIoQueueDispatchSequential
        );

      // register event callbacks
    queueConfig.EvtIoRead   = MemReaderEvtIoRead;
    queueConfig.EvtIoWrite  = MemReaderEvtIoWrite;

    // Fill in a callback for destroy, and our QUEUE_CONTEXT size
    WDF_OBJECT_ATTRIBUTES_INIT_CONTEXT_TYPE(&queueAttributes,
QUEUE_CONTEXT);
    queueAttributes.EvtDestroyCallback =
MemReaderEvtIoQueueContextDestroy;

    status = WdfIoQueueCreate(
                 Device,
                 &queueConfig,
                 &queueAttributes,
                 &queue
                 );

    if( !NT_SUCCESS(status) ) {
        DbgPrint("WdfIoQueueCreate failed 0x%x\n",status);
        return status;
    }

    // Get our Driver Context memory from the returned Queue handle
    queueContext = QueueGetContext(queue);

      //initialise queue context
    queueContext->CurrentRequest    = NULL;
      queueContext->ma.len                = 0;
      queueContext->ma.offset             = 0;
      queueContext->hPhysicalMemory = NULL;
      queueContext->pVirtualAddr          = NULL;

      //Initializes a counted Unicode string
      RtlInitUnicodeString( &usPhysicalMemory,
L"\\Device\\PhysicalMemory");
```

```
      //The InitializeObjectAttributes macro initializes the opaque
OBJECT_ATTRIBUTES
      //structure, which specifies the properties of an object handle
to routines that
      //open handles.
      //The OBJECT_ATTRIBUTES structure specifies attributes that can
be applied to
      //objects or object handles by routines that create objects
and/or return handles
      //to objects.
      InitializeObjectAttributes( &oa, &usPhysicalMemory,
                                          OBJ_CASE_INSENSITIVE |
OBJ_KERNEL_HANDLE,
                                          (HANDLE) NULL,
                                          (PSECURITY_DESCRIPTOR)
NULL);

      //The ZwOpenSection routine opens a handle for an existing
section object.
      //An object that represents a section of memory that can be
shared
      status = ZwOpenSection( &(queueContext->hPhysicalMemory),
SECTION_ALL_ACCESS, &oa );
      if( !NT_SUCCESS(status) )
      {
            DbgPrint("MEM: Failed to open %wZ! Status %p\n",
usPhysicalMemory, status);
            queueContext->hPhysicalMemory = NULL;
            return status;
      }
      DbgPrint("MEM: Opened section handle %p in driver\n",
queueContext->hPhysicalMemory);

    return status;
}

VOID
MemReaderEvtIoQueueContextDestroy(
    WDFOBJECT Object
)
{
      DbgPrint("MEM: IoContextQueueDestroy!\n");
    return;
}


VOID
MemReaderEvtRequestCancel(
    IN WDFREQUEST Request
    )
{
    PQUEUE_CONTEXT queueContext =
QueueGetContext(WdfRequestGetIoQueue(Request));
```

```
      DbgPrint("MEM: MemReaderRequestCancel called on Request 0x%p\n",
Request);

    WdfRequestCompleteWithInformation(Request, STATUS_CANCELLED, 0L);

    // This book keeping is synchronized by the common
    // Queue presentation lock
    ASSERT(queueContext->CurrentRequest == Request);
    queueContext->CurrentRequest = NULL;

    return;
}

VOID
MemReaderEvtIoRead(
    IN WDFQUEUE   Queue,
    IN WDFREQUEST Request,
    IN size_t     Length
    )
{
    NTSTATUS            Status;
    WDFMEMORY           memory;
      LARGE_INTEGER     sectionOffset ;
    PQUEUE_CONTEXT      queueContext      = QueueGetContext(Queue);
      ULONG             writeBlockSize    = TX_BLOCK_SIZE;

      DbgPrint("MEM: EchoEvtIoRead Called! Queue 0x%p, Request 0x%p
Length %d, writeBlockSize %d,\
                  Offset
%d\n",Queue,Request,Length,writeBlockSize,queueContext->ma.offset);

      // Read what we have
    if( Length <= 0 ) {
        WdfRequestCompleteWithInformation(Request, STATUS_SUCCESS,
(ULONG_PTR)0L);
        return;
    }

      if( writeBlockSize > (ULONG)(queueContext->ma.len - queueContext-
>ma.offset) ){
            writeBlockSize = (ULONG)(queueContext->ma.len -
queueContext->ma.offset);
      }

      DbgPrint("MEM: After decrement ma.len=%u\n", queueContext-
>ma.len);
      sectionOffset.QuadPart = queueContext->ma.offset;
      Status = ZwMapViewOfSection( queueContext->hPhysicalMemory, //
handle to section object
            ZwCurrentProcess(), // macro to retrieve handle for current
process
            &(queueContext->pVirtualAddr),
            0L, // ZeroBits
            Length, // CommitSize
            &sectionOffset, // Section Offset
            &Length, // View Size
            ViewShare, // Inherit Disposition
            0,
            PAGE_READONLY // Win32 Protection
            );
      //increase offset even if mapping fails
      queueContext->ma.offset += writeBlockSize;

    if( !NT_SUCCESS(Status)) {
            DbgPrint("MEM: EvtIoRead Mapping of Section Object failed
Status: 0x%x\n",Status);
            DbgPrint("MEM: EvtIoRead Mapping of Section Object failed
ma.offset=%u ma.len=%u\n",
                  (queueContext->ma.offset), queueContext->ma.len);
        WdfVerifierDbgBreakPoint();
        WdfRequestCompleteWithInformation(Request, Status, 0L);
        return;
    }

    // Get the request memory
    Status = WdfRequestRetrieveOutputMemory(Request, &memory);
    if( !NT_SUCCESS(Status) ) {
            DbgPrint("MEM: EvtIoRead Could not get request memory
buffer 0x%x\n",Status);
        WdfVerifierDbgBreakPoint();
        WdfRequestCompleteWithInformation(Request, Status, 0L);
        return;
    }

    // Copy the memory out
      DbgPrint("MEM: Copy %d bytes from buffer!\n", Length);
    Status = WdfMemoryCopyFromBuffer( memory, // destination
                             0,      // offset into the destination
memory
                             queueContext->pVirtualAddr,//
queueContext->Buffer
                             (Length) );
    if( !NT_SUCCESS(Status) ) {
            DbgPrint("MEM: EvtIoRead: WdfMemoryCopyFromBuffer failed
0x%x\n", Status);
        WdfRequestComplete(Request, Status);
        return;
    }

      if(queueContext->pVirtualAddr){
            DbgPrint("MEM: EvtIoRead Unmapping Section Object!\n");
            Status = ZwUnmapViewOfSection(
                  ZwCurrentProcess(), // macro to retrieve handle for
current process
                  queueContext->pVirtualAddr) ;
            queueContext->pVirtualAddr = NULL;

            if( !NT_SUCCESS(Status)) {
                  DbgPrint("MEM: EvtIoRead Unmapping of Section Object
failed: 0x%x\n",Status);
```

```
                  WdfVerifierDbgBreakPoint();
                  WdfRequestCompleteWithInformation(Request, Status,
0L);
                  return;
            }
      }

    // Set transfer information
    WdfRequestSetInformation(Request, (ULONG_PTR)(Length));

    // Mark the request is cancelable
    WdfRequestMarkCancelable(Request, MemReaderEvtRequestCancel);

      WdfRequestComplete(Request, Status);
      DbgPrint("MEM: Completed EvtIoRead View Size=%d!\n",Length);
    return;
}

VOID
MemReaderEvtIoWrite(
    IN WDFQUEUE   Queue,
    IN WDFREQUEST Request,
    IN size_t     Length
    )
{
    NTSTATUS            Status;
    WDFMEMORY           memory;
    PQUEUE_CONTEXT      queueContext = QueueGetContext(Queue);
      PVOID             temp = ExAllocatePoolWithTag(NonPagedPool,
Length, 'sam1');

      DbgPrint("MEM: EvtIoWrite Called! Queue 0x%p, Request 0x%p Length
%d\n",
                            Queue,Request,Length);

      // Get the memory buffer
    Status = WdfRequestRetrieveInputMemory(Request, &memory);
    if( !NT_SUCCESS(Status) ) {
            DbgPrint("MEM: EvtIoWrite Could not get request memory
buffer 0x%x\n",
                 Status);
        WdfVerifierDbgBreakPoint();
        WdfRequestComplete(Request, Status);
        return;
    }

    // Copy the memory in
    Status = WdfMemoryCopyToBuffer( memory,
                           0,  // offset into the source memory
                           temp,
                           Length );

    if( !NT_SUCCESS(Status) ) {
            DbgPrint("MEM: EchoEvtIoWrite WdfMemoryCopyToBuffer failed
0x%x\n", Status);
```

```
        WdfVerifierDbgBreakPoint();
        WdfRequestComplete(Request, Status);
        return;
    }

      queueContext->ma.len = ((memaddr *)temp)->len;
      queueContext->ma.offset = ((memaddr *)temp)->offset;
      DbgPrint("MEM: ma.len %u ma.offset %u!\n",queueContext-
>ma.len,queueContext->ma.offset);

      // Set transfer information
    WdfRequestSetInformation(Request, (ULONG_PTR)Length);
      DbgPrint("MEM: Write After Set Information!\n");

    // Specify the request is cancelable
    WdfRequestMarkCancelable(Request, MemReaderEvtRequestCancel);
      DbgPrint("MEM: Write After RequestMarkCancellable!\n");

    queueContext->CurrentRequest = Request;

      WdfRequestComplete(Request, Status);
      DbgPrint("MEM: return!\n");

    return;
}
```

memreaderapp.cpp

```
/*
* Copyright (c) 2008, Vincent Cheong
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
are met:
*     * Redistributions of source code must retain the above copyright
*       notice, this list of conditions and the following disclaimer.
*     * Redistributions in binary form must reproduce the above
copyright
*       notice, this list of conditions and the following disclaimer in
the
*       documentation and/or other materials provided with the
distribution.
*     * Neither the name of the Naval Postgraduate School nor the
*       names of its contributors may be used to endorse or promote
products
*       derived from this software without specific prior written
permission.
*
* THIS SOFTWARE IS PROVIDED BY VINCENT CHEONG ''AS IS'' AND ANY
* EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED
* WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE
* DISCLAIMED. IN NO EVENT SHALL <copyright holder> BE LIABLE FOR ANY
* DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES
* (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES;
* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND
* ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
TORT
* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS
* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

#define INITGUID

#include <windows.h>
#include <strsafe.h>
#include <setupapi.h>
#include <stdio.h>
#include <stdlib.h>
#include "public.h"

#define NUM_ASYNCH_IO   100

#define VISTA_VERSION_NUMBER  6
#define XP_VERSION_NUMBER          5
```

```
#define ACCEPTABLE_VERSION          XP_VERSION_NUMBER

#define READER_TYPE   1
#define WRITER_TYPE   2
typedef struct _memaddr{
    ULONGLONG offset;
    ULONGLONG len;
} memaddr;


PCHAR DevicePath;
memaddr ma;

PCHAR
GetDevicePath(
    IN  LPGUID InterfaceGuid
    );

BOOLEAN
correctOsVersion();

ULONGLONG
CheckDiskSpace(
      IN HANDLE hDumpFile
      );

BOOLEAN
WriteMemAddr(
    IN HANDLE hDevice
                   );

BOOLEAN
ReadRam(
    IN HANDLE hDevice,
    IN HANDLE hDumpFile,
      IN ULONG  blockSize
            );

void __cdecl
main(
    __in int argc,
    __in_ecount(argc) char* argv[]
    )
{
    HANDLE  hDevice           = INVALID_HANDLE_VALUE,
                  hDumpFile   = INVALID_HANDLE_VALUE;
    HANDLE  th1 = NULL;
      ULONGLONG   testLength = 0;
    BOOLEAN result;
      ULONG offset = 0 ;

      if( argc == 2 ) {
      }
      else if ( argc == 3 ){
            offset = atol(argv[2]) * 1024 * 1024; // in MB
      }
      else{
            printf("Wrong Usage:    > memreader <filename>\n");
            printf("Wrong Usage: or > memreader <filename> <offset in
MB> <end>\n");
            goto exit;
      }

      printf("Dumping to file %s\n", argv[1]);
      printf("Enter 'q' to exit and any key to continue: ");

      int ip = getchar();

      if( (ip == 'q') || (ip == 'Q') ){
            goto exit;
      }
      else {
      }

      ma.offset = offset;
      ma.len = 0;

      if( !correctOsVersion() )
            goto exit ;


      WCHAR dumpFile[MAX_PATH] = { L'\0' };

      // just do a simple file name check and we are good to go?
      hDumpFile = CreateFile( argv[1], GENERIC_WRITE, 0, NULL,
CREATE_ALWAYS, 0, NULL );
      if( hDumpFile == INVALID_HANDLE_VALUE ) {
            fprintf( stderr, " -> ERROR: Failed to open dump file
\"%s\"! (%u)\n\n", dumpFile, GetLastError());
            printf(" -> ERROR: Failed to open dump file \"%s\"!
(%u)\n\n", dumpFile, GetLastError());
            goto exit;
      }

      testLength = CheckDiskSpace(hDumpFile);

    DevicePath = GetDevicePath((LPGUID)&GUID_DEVINTERFACE_ECHO);

    printf("DevicePath: %s\n", DevicePath);

    hDevice = CreateFile(DevicePath,
                         GENERIC_READ|GENERIC_WRITE,
                         FILE_SHARE_READ,
                         NULL,
                         OPEN_EXISTING,
                         0,
                         NULL );

    if (hDevice == INVALID_HANDLE_VALUE) {
        printf("Failed to open device. Error %d\n",GetLastError());
        goto exit;
    }

    printf("Opened device successfully\n");
      WriteMemAddr(hDevice);
      Sleep(100);
      ReadRam(hDevice, hDumpFile, TX_BLOCK_SIZE);

exit:
    if (th1 != NULL) {
        WaitForSingleObject(th1, INFINITE);
        CloseHandle(th1);
    }

    if (hDevice != INVALID_HANDLE_VALUE) {
        CloseHandle(hDevice);
    }
      printf("Exiting...\n");
    return;

}

PUCHAR
CreatePatternBuffer(
    IN ULONG Length
    )
{
    unsigned int i;
    PUCHAR p, pBuf;

    pBuf = (PUCHAR)malloc(Length);
    if( pBuf == NULL ) {
        printf("Could not allocate %d byte buffer\n",Length);
        return NULL;
    }

    p = pBuf;

    for(i=0; i < Length; i++ ) {
        *p = (UCHAR)i;
        p++;
    }

    return pBuf;
}

BOOLEAN
VerifyPatternBuffer(
    IN PUCHAR pBuffer,
    IN ULONG  Length
    )
{
    unsigned int i;
    PUCHAR p = pBuffer;
    for( i=0; i < Length; i++ ) {

        if( *p != (UCHAR)(i & 0xFF) ) {
            printf("Pattern changed. SB 0x%x, Is 0x%x\n",
                   (UCHAR)(i & 0xFF), *p);
            return FALSE;
        }

        p++;
    }

    return TRUE;
}


PCHAR
GetDevicePath(
    IN  LPGUID InterfaceGuid
    )
{
    HDEVINFO HardwareDeviceInfo;
    SP_DEVICE_INTERFACE_DATA DeviceInterfaceData;
    PSP_DEVICE_INTERFACE_DETAIL_DATA DeviceInterfaceDetailData = NULL;
    ULONG Length, RequiredLength = 0;
    BOOL bResult;

    HardwareDeviceInfo = SetupDiGetClassDevs(
                             InterfaceGuid,
                             NULL,
                             NULL,
                             (DIGCF_PRESENT | DIGCF_DEVICEINTERFACE));

    if (HardwareDeviceInfo == INVALID_HANDLE_VALUE) {
        printf("SetupDiGetClassDevs failed!\n");
        exit(1);
    }

    DeviceInterfaceData.cbSize = sizeof(SP_DEVICE_INTERFACE_DATA);

    bResult = SetupDiEnumDeviceInterfaces(HardwareDeviceInfo,
                                              0,
                                              InterfaceGuid,
                                              0,
                                              &DeviceInterfaceData);

    if (bResult == FALSE) {

        LPVOID lpMsgBuf;

        if (FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER |
                          FORMAT_MESSAGE_FROM_SYSTEM |
                          FORMAT_MESSAGE_IGNORE_INSERTS,
                          NULL,
                          GetLastError(),
                          MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
                          (LPSTR) &lpMsgBuf,
                          0,
                          NULL
                          )) {

            printf("Error: %s", (LPSTR)lpMsgBuf);
            LocalFree(lpMsgBuf);
        }

        printf("SetupDiEnumDeviceInterfaces failed.\n");
        SetupDiDestroyDeviceInfoList(HardwareDeviceInfo);
        exit(1);
    }

    SetupDiGetDeviceInterfaceDetail(
        HardwareDeviceInfo,
        &DeviceInterfaceData,
        NULL,
        0,
        &RequiredLength,
        NULL
        );

    DeviceInterfaceDetailData = (PSP_DEVICE_INTERFACE_DETAIL_DATA)
LocalAlloc(LMEM_FIXED, RequiredLength);

    if (DeviceInterfaceDetailData == NULL) {
        SetupDiDestroyDeviceInfoList(HardwareDeviceInfo);
        printf("Failed to allocate memory.\n");
        exit(1);
    }

    DeviceInterfaceDetailData->cbSize =
sizeof(SP_DEVICE_INTERFACE_DETAIL_DATA);

    Length = RequiredLength;

    bResult = SetupDiGetDeviceInterfaceDetail(
                  HardwareDeviceInfo,
                  &DeviceInterfaceData,
                  DeviceInterfaceDetailData,
                  Length,
                  &RequiredLength,
                  NULL);

    if (bResult == FALSE) {

        LPVOID lpMsgBuf;

        if (FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER |
                      FORMAT_MESSAGE_FROM_SYSTEM |
                      FORMAT_MESSAGE_IGNORE_INSERTS,
                      NULL,
                      GetLastError(),
                      MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
                      (LPSTR) &lpMsgBuf,
                      0,
                      NULL
                      )) {

            MessageBox(NULL, (LPCTSTR) lpMsgBuf, "Error", MB_OK);
            LocalFree(lpMsgBuf);
        }
        printf("Error in SetupDiGetDeviceInterfaceDetail\n");

        SetupDiDestroyDeviceInfoList(HardwareDeviceInfo);
        LocalFree(DeviceInterfaceDetailData);
        exit(1);
    }

    return DeviceInterfaceDetailData->DevicePath;

}



BOOLEAN
correctOsVersion()
{
      OSVERSIONINFO     osvi;
      ZeroMemory(&osvi, sizeof(OSVERSIONINFO));

      osvi.dwOSVersionInfoSize = sizeof(OSVERSIONINFO);

      if(!GetVersionEx ((OSVERSIONINFO *) &osvi))
            return FALSE;

      if ( VER_PLATFORM_WIN32_NT==osvi.dwPlatformId &&
            osvi.dwMajorVersion > ACCEPTABLE_VERSION )
      {
            if(osvi.dwMajorVersion == VISTA_VERSION_NUMBER )
                  printf("Windows VISTA detected\n");
            else if (osvi.dwMajorVersion == XP_VERSION_NUMBER )
                  printf("Pre VISTA Version of Windows detected\n");
            else
                  printf("Post VISTA Version(%d) of Windows detected, \
                  application has not been tested for this version.\n",
osvi.dwMajorVersion);
            return TRUE;
      }

      return FALSE;
}

ULONGLONG
CheckDiskSpace(
      IN HANDLE hDumpFile
      )
{
      // Check that there is sufficient space the full memory dump
```

```
	MEMORYSTATUSEX    ms;
	ms.dwLength = sizeof(MEMORYSTATUSEX);
	//Retrieves information about the system's current usage of both 
physical and virtual memory.
	if( !GlobalMemoryStatusEx( &ms )) {
		fprintf( stderr, " -> ERROR: GlobalMemoryStatusEx failed, 
%u\n", GetLastError());
		return ma.len;
	}

	LARGE_INTEGER liDistanceToMove;
	liDistanceToMove.QuadPart = ms.ullTotalPhys;
	//Sets the physical file size for the specified file to the 
current position of the file pointer.
	ma.len = (ULONGLONG)ms.ullTotalPhys;
	printf("File Size is:%f\n",(float)ma.len/(1024*1024));

	//uncomment later
	//SetFilePointerEx(hDumpFile, liDistanceToMove, NULL, 
FILE_BEGIN);
	if (!SetEndOfFile(hDumpFile))
	{
		fprintf( stderr, " -> ERROR: Not enough room to store 
memory dump\n");
		return ma.len;
	}
	//Move the file pointer back to the beginning of the file.
	liDistanceToMove.QuadPart = 0;
	SetFilePointerEx(hDumpFile, liDistanceToMove, NULL, FILE_BEGIN);

	return ma.len;
}

BOOLEAN
WriteMemAddr(
    IN HANDLE hDevice
                  )
{
	BOOLEAN result = TRUE;
	ULONG retLen = 16 ;
	printf("Writing Mem Addr to driver.\n");
	if(!WriteFile(hDevice, &ma, sizeof(memaddr), &retLen, NULL)){
		printf("ERROR: Write Mem Addr failed, %u\n", 
GetLastError());
		result = FALSE;
	} else {
		printf("Mem Addr successfully written to driver!\n");
	}


	return result;
}

BOOLEAN
ReadRam(
    IN HANDLE hDevice,
    IN HANDLE hDumpFile,
      IN ULONG  blockSize
            )
{
    ULONGLONG     bytesToRead       = blockSize,
                        offset                  = ma.offset;
      ULONG       blocksWritten     = 0;

    BOOLEAN       result                  = TRUE;
      bool        loop              = true;
      PUCHAR            tempDataHolder    = (PUCHAR)malloc(blockSize);
      BYTE        zeros[TX_BLOCK_SIZE]= {0};
      int               iterations        = 0,
                        goods             = 0,
                        bads              = 0;

//    while( loop )
      while((offset!=ma.len))
      {
            if( offset + bytesToRead >= ma.len){
                  bytesToRead = ma.len - offset;
                  loop = false;
            }
            offset += bytesToRead;
            printf("%d: ( offset %u <",++iterations,offset);
            printf(" ma.len %u )", ma.len);
            printf(" == %d\n",(offset != ma.len));
            memset(tempDataHolder, 0, blockSize);
            if ( !ReadFile (hDevice, // file to read
                        tempDataHolder, // buffer to receive data
                        blockSize, // number of bytes to read in total
                        &blocksWritten, // number of bytes read
                        NULL)) {
                  bads++;
                  printf ("READRAM: ReadFile failed: Error %d\n",
GetLastError());
                  //result = FALSE;
                  //goto Cleanup;

            } else {
                  //printf("Bytes returned is :%d!\n", blocksWritten);
                  //printf("ma.len is :%u!\n", ma.len);
                  goods++;
            }
            BOOL writeSuccess =     WriteFile( hDumpFile,
tempDataHolder,
                  blockSize, &blocksWritten, NULL);
            //printf("Bytes written is :%d!\n", blocksWritten);
            if( !writeSuccess )     {
                  printf(" -> ERROR: WriteFile failed, %u\n",
GetLastError());
                  result = FALSE;
                  goto Cleanup;
            }
            if(offset >= ma.len){
                  printf("Completed with goods=%d, bads=%d!\n", goods,
bads);
                  goto Cleanup;
            }
      }

Cleanup:

      if( hDumpFile )
            CloseHandle( hDumpFile );

    return result;
}
```

memreader.inx

```
[Version]
Signature="$WINDOWS NT$"
Class=Sample
ClassGuid={78A1C341-4539-11d3-B88D-00C04FAD5171}
Provider=%MSFT%
DriverVer=03/20/2003,5.00.3788
CatalogFile=KmdfSamples.cat

[DestinationDirs]
DefaultDestDir = 12

; ================= Class section =====================

[ClassInstall32]
Addreg=SampleClassReg

[SampleClassReg]
HKR,,,0,%ClassName%
HKR,,Icon,,-5

[SourceDisksNames]
1 = %DiskId1%,,,""

[SourceDisksFiles]
memreader.sys  = 1,,

;*****************************************
; MemReader  Install Section
;*****************************************

[Manufacturer]
%StdMfg%=Standard,NT$ARCH$

; Following section is meant for Windows 2000 as it
; cannot parse decorated model sections
[Standard]
;
; Hw Id is root\MemReader
;
%MemReader.DeviceDesc%=MemReader_Device, root\MemReader

; Decorated model section take precedence over undecorated
; ones on XP and later.
[Standard.NT$ARCH$]
%MemReader.DeviceDesc%=MemReader_Device, root\MemReader

[MemReader_Device.NT]
CopyFiles=Drivers_Dir

[Drivers_Dir]
memreader.sys
```

```
;------------- Service installation
[MemReader_Device.NT.Services]
AddService = MemReader,%SPSVCINST_ASSOCSERVICE%, MemReader_Service_Inst

; ------------- MemReader driver install sections
[MemReader_Service_Inst]
DisplayName    = %MemReader.SVCDESC%
ServiceType    = 1               ; SERVICE_KERNEL_DRIVER
StartType      = 3               ; SERVICE_DEMAND_START
ErrorControl   = 1               ; SERVICE_ERROR_NORMAL
ServiceBinary  = %12%\memreader.sys
LoadOrderGroup = Extended Base

;
;--- MemReader_Device Coinstaller installation ------
;

[DestinationDirs]
MemReader_Device_CoInstaller_CopyFiles = 11

[MemReader_Device.NT.CoInstallers]
AddReg=MemReader_Device_CoInstaller_AddReg
CopyFiles=MemReader_Device_CoInstaller_CopyFiles

[MemReader_Device_CoInstaller_AddReg]
HKR,,CoInstallers32,0x00010000,
"wdfcoinstaller01005.dll,WdfCoInstaller"

[MemReader_Device_CoInstaller_CopyFiles]
wdfcoinstaller01005.dll

[SourceDisksFiles]
wdfcoinstaller01005.dll=1 ; make sure the number matches with
SourceDisksNames

[MemReader_Device.NT.Wdf]
KmdfService =  memreader, memreader_wdfsect
[MemReader_wdfsect]
KmdfLibraryVersion = 1.5


[Strings]
SPSVCINST_ASSOCSERVICE= 0x00000002
MSFT = "Microsoft"
StdMfg = "(Standard system devices)"
DiskId1 = "Memory Reader Installation Disk #1"
MemReader.DeviceDesc = "Memory Reader Service Driver"
MemReader.SVCDESC = "Memory Reader Service"
ClassName       = "Tools"
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten, Lest We Remember: Cold Boot Attacks on Encryption Keys, Proc. 2008 USENIX Security Symposium.

[2] Mariusz Burdach, Physical Memory Forensics, Black Hat 2006

[3] Timothy Vidas, The Acquisition and Analysis of Random Access Memory, Journal of Digital Forensic Practice, Volume 1 Issue 4 December 2006.

[4] Jesse D. Kornblum, Exploiting the Rootkit Paradox with Windows Memory Analysis, International Journal of Digital Evidence Fall 2006.

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

[6] Adam Boileau, Hit by a Bus: Physical Access Attacks with Firewire, Ruxcon 2006.

[7] George Garner, Forensic Acquisition Utilities http://www.gmgsystemsinc.com/fau/, date last accessed Nov 2008.

[8] Helix 3 – Incident Response, Computer Forensics, Electronic Discovery, http://www.e-fense.com/helix/, date last accessed Nov 2008.

[9] Gerald J Popek, Robert P Goldberg, Formal Requirements for Virtualizable Third Generation Architectures, Communications of the ACM July 1974.

[10] VMware, http://www.vmware.com, date last accessed Nov 2008.

[11] 010 Editor, http://www.sweetscape.com/010editor/, SweetScape software, date last accessed Nov 2008.

[12] Visual Binary Diff, http://www.cjmweb.net/vbindiff/, Christopher J. Madsen, date last accessed Nov 2008.

[13] Microsoft Connect, https://connect.microsoft.com/, date last accessed Nov 2008.

[14] Windows Hardware Developer Central (WHDC), http://www.microsoft.com/whdc/, date last accessed Nov 2008.

[15] ManTech International Corporation, http://www.mantech.com/mantech.asp, date last accessed Nov 2008.

[16] ManTech Memory DD, http://www.mantech.com/msma/mdd.asp, date last accessed Nov 2008.

[17] WHDC, User-Mode Driver Framework (UMDF), http://www.microsoft.com/whdc/driver/wdf/UMDF.mspx, date last accessed Nov 2008.

[18] WHDC, Kernel-Mode Driver Framework (KMDF), http://www.microsoft.com/whdc/driver/wdf/KMDF.mspx, date last accessed Nov 2008.

[19] MSDN, INF Add Service Directive, http://msdn.microsoft.com/en-us/library/ms794559.aspx, date last accessed Nov 2008.

[20] MSDN, Create a Service Function, http://msdn.microsoft.com/en-us/library/ms682450(VS.85).aspx, date last accessed Nov 200.

[21] MSDN, Providing a Driver Package, http://msdn.microsoft.com/en-us/library/ms791680.aspx, date last accessed Nov 2008.

[22] Installing Debugging Tools for Windows 32-bit Version, http://www.microsoft.com/whdc/devtools/debugging/installx86.mspx, date last accessed Nov 2008.

[23] DebugView for Windows, http://technet.microsoft.com/en-us/sysinternals/bb896647.aspx, date last accessed Nov 2008.

[24] MSDN, ZwOpenSection, http://msdn.microsoft.com/en-us/library/ms804353.aspx, date last accessed Nov 2008.

[25] Microsoft Technet, Device\PhysicalMemory location Object, http://technet.microsoft.com/en-us/library/cc787565.aspx, date last accessed Nov 2008.

[26] MSDN, DbgPrint, http://msdn.microsoft.com/en-us/library/ms792790.aspx, date last accessed Nov 2008.

[27] Operating System Market Share, http://marketshare.hitslink.com/report.aspx?qprid=8, date last accessed Nov 2008.

[28] MSDN, ZwMapViewOfSection, http://msdn.microsoft.com/en-us/library/ms804369.aspx, date last accessed Nov 2008.

[29] MSDN, Using NTSTATUS values, http://msdn.microsoft.com/en-us/library/aa489609.aspx, date last accessed Nov 2008.

[30] NTSTATUS codes, http://nologs.com/ntstatus.html, date last accessed Nov 2008.

[31] WHDC, Introduction to the Windows Driver Foundation, http://www.microsoft.com/whdc/driver/wdf/wdf-intro.mspx, date last accessed Nov 2008.

[32] Microsoft Developer Network, http://msdn.microsoft.com/en-us/default.aspx, date last accessed Nov 2008.

[33] Microsoft Technet, Demystifying the 'Blue Screen of Death', http://technet.microsoft.com/en-us/library/cc750081.aspx, date last accessed Nov 2008.

[34] MSDN, Bug Check Codes, http://msdn.microsoft.com/en-us/library/ms789516.aspx, date last accessed Nov 2008.

[35] Digital Signatures for Kernel Modules on Systems Running Windows Vista, http://www.microsoft.com/whdc/winlogo/drvsign/kmsigning.mspx, date last accessed Nov 2008.

[36] Archictecture of the Kernel-Mode Driver Framework, http://www.microsoft.com/whdc/driver/wdf/KMDF-arch.mspx, date last accessed Nov 2008.

[37] MSDN, Creating an INF File for a File System Filter Driver, http://msdn.microsoft.com/en-us/library/ms790727.aspx, date last accessed Nov 2008.

[38] MSDN, System Error Codes 90-499), http://msdn.microsoft.com/en-us/library/ms681382(VS.85).aspx, date last accessed Nov 2008.

[39] WHDC, Windows Driver Kit (WDK) – Overview, http://www.microsoft.com/whdc/devtools/wdk/default.mspx, date last accessed Nov 2008.

[40] MSDN, Certificate Creation Tool (Makecert.exe), http://msdn.microsoft.com/en-us/library/bfsktky3(VS.80).aspx, date last accessed Nov 2008.

[41] WHDC, Microsoft Cross-certificates for Windows Vista Kernel Mode Code Signing, http://www.microsoft.com/whdc/winlogo/drvsign/crosscert.mspx, date last accessed Nov 2008.

[42] MSDN, GlobalMemoryStatusEx Function, http://msdn.microsoft.com/en-us/library/aa366589(VS.85).aspx, date last accessed Nov 2008.

[43] Forensics Wiki, Dd, http://www.forensicswiki.org/index.php?title=Dd, Paul Rubin, Davod MacKenzie, Stuart Kemp, date last accessed Nov 2008.

[44] WHDC, Download Windows Symbol Package, http://www.microsoft.com/whdc/devtools/debugging/symbolpkg.mspx, date last accessed Nov 2008.

[45] Arne Vidstrum, Forensic Ram Dumping, http://www.bandwidthco.com/whitepapers/compforensics/memory/Forensic%20RAM%20Dumping.pdf, last accessed on Nov 2008.

[46] MSDN, Driver Entry, http://msdn.microsoft.com/en-us/library/ms795702.aspx, last accessed on Nov 2008.

[47] MSDN, WDF_DRIVER_CONFIG_INIT, http://msdn.microsoft.com/en-us/library/aa491341.aspx, last accessed on Nov 2008.

[48] MSDN, WdfDriverCreate, http://msdn.microsoft.com/en-us/library/aa491318.aspx, last accessed on Nov 2008.

[49] MSDN, WdfDeviceCreate, http://msdn.microsoft.com/en-us/library/aa491109.aspx, last accessed on Nov 2008.

[50] MSDN, WdfDeviceCreateDeviceInterface, http://msdn.microsoft.com/en-us/library/aa490963.aspx, last accessed on Nov 2008.

[51] MSDN, WdfIoQueueCreate, http://msdn.microsoft.com/en-us/library/aa491614.aspx, last accessed on Nov 2008.

[52] MSDN, WdfRequestRetrieveOutputMemory, http://msdn.microsoft.com/en-us/library/aa492080.aspx, last accessed on Nov 2008.

[53] MSDN, WdfMemoryCopyFromBuffer, http://msdn.microsoft.com/en-us/library/aa491569.aspx, last accessed on Nov 2008.

[54] MSDN, WdfRequestCompleteWithInformation, http://msdn.microsoft.com/en-us/library/aa492029.aspx, last accessed on Nov 2008.

[55] MSDN, WdfMemoryCopyToBuffer, http://msdn.microsoft.com/en-us/library/aa491573.aspx, last accessed on Nov 2008.

[56] MSDN, WdfRequestRetrieveInputBuffer, http://msdn.microsoft.com/en-us/library/aa492051.aspx, last accessed on Nov 2008.

[57] MSDN, Enabling Test-signing, http://msdn.microsoft.com/en-us/library/aa906238.aspx, last accessed on Nov 2008.

[58] OSR Online, DDK Build – Visual Studio .CMD Procedure For Building Drivers, http://www.osronline.com/article.cfm?name=ddkbuild_v73r27.zip&id=43, last accessed on Nov 2008.

[59] Hollis Technology Solutions, http://www.hollistech.com/, last accessed on Nov 2008.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Timothy Vidas
   Naval Postgraduate School
   Monterey, California

4. Dr. George Dinolt
   Naval Postgraduate School
   Monterey, California

5. Cheong Choong Wee Vincent
   Naval Postgraduate School
   Monterey, California

6. Professor Yeo Tat Soon, Director
   Temasek Defence Systems Institute
   National University of Singapore
   Singapore

7. Tan Lai Poh (Ms), Assistant Manager
   Temasek Defence Systems Institute
   National University of Singapore
   Singapore