May 28th, 4:40 PM

Applying Memory Forensics to Rootkit Detection

Igor Korkin
National Research Nuclear University, Moscow Engineering Physics Institute, igor.korkin@gmail.com

Ivan Nesterov
Moscow Institute of Physics and Technology, i.nesterow@gmail.com

(c)ADFSL

APPLYING MEMORY FORENSICS TO ROOTKIT DETECTION

Igor Korkin
National Research Nuclear University
Moscow Engineering Physics Institute (NRNU MEPhI)
Moscow, 115409, Russia
igor.korkin@gmail.com

Ivan Nesterov
Moscow Institute of Physics and Technology (MIPT)
Moscow Region 141700, Russia
i.nesterow@gmail.com

ABSTRACT

Volatile memory dump and its analysis is an essential part of digital forensics. Among a number of various software and hardware approaches for memory dumping there are authors who point out that some of these approaches are not resilient to various anti-forensic techniques, and others that require a reboot or are highly platform dependent. New resilient tools have certain disadvantages such as low speed or vulnerability to rootkits which directly manipulate kernel structures, e.g., page tables. A new memory forensic system – Malware Analysis System for Hidden Knotty Anomalies (MASHKA) is described in this paper. It is resilient to popular anti-forensic techniques. The system can be used for doing a wide range of memory forensics tasks. This paper describes how to apply the system for research and detection of kernel mode rootkits and also presents analysis of the most popular anti-rootkit tools.

Keywords: Digital forensics, Virtual memory acquisition, Malware research, Rootkits detection, Anti-forensics.

1. INTRODUCTION

Memory dump is used in various aspects of information security. It can be used for controlling virtual memory content while program is executed, running and after its close, is also typical for sophisticated malware, reverse-engineering due to it provides code and data in virtual memory for research and analysis. Memory dump is also used in computer forensic examination processes.

A fairly common problem is to obtain and analyze a memory dump. Both individual professionals J. Stuttgen, M. Cohen, B. Schatz, J. Okolica, J. Rutkowska, J. Butler, L. Cavallaro, L. Milkovich and entire international companies such as Microsoft, WindowsSCOPE, Guidance Software, Mandiant Corporation, Volatile Systems LLC tried to deal with this problem. A number of research theses are devoted to these issues.

It has also been discussed during various international conferences like BlackHat, DefCon, Digital Forensic Research Workgroup (DFRWS) Conference, ADFSL Conference on Digital Forensics, Security and Law, Open Source Digital Forensics Conference and workshops such as International Workshop on Digital Forensics (WSDF), SANS Windows Memory Forensics Training (FOR526), Open Memory Forensics Workshop (OMFW) by Volatile Systems.

This article presents a new memory dumping and analysis system which has several advantages and gives an example of how to use it for the kernel-mode rootkits and hidden malware detection. Moreover, this system can be applied in all mentioned above areas. The remainder of the paper is organized as follows.

Section 2 is devoted to the most popular software and hardware approaches for acquiring memory their analysis, including a new low-level approach. Memory dump can be obtained by executing a code that is running in user mode, kernel mode, VMX-root mode, system management mode and low-level AMT code which is used by an independent processor. These approaches can dump memory of single process address space or copy physical Random Access Memory (RAM). Tools and approaches focused on the mentioned code modes are described. As Microsoft Windows operating system is the most popular now it is essential to focus on OS Windows family of tools. However, similar conclusions could be made about Unix-based tools and approaches.

Section 3 contains a description of author's memory dump acquisition approach. The idea is based on walking through the page tables and saving each of them with additional information, such as virtual page addresses and its offsets in the result dump file. This approach reveals good efficiency when each page is not separately saved to HDD, but is buffered and archived before it is saved. Additional dump file encryption protects it from modification while it is being saved to HDD. This approach uses memory paging in protected mode and therefore is operating system independent and is applicable on Linux or Mac OS X.

In Section 4 hidden malware is observed. The current available detection methods and tools are analyzed with the focus on signature detection of hidden drivers as the most common problem. An author's Dynamic Bit Signature (DBS) and Rating Point Inspection (RPI) approaches for processes' and drivers' detection and comparative analysis are briefly presented.

Section 5 contains main conclusions and further research directions.

2. RELATED WORK

2.1 Virtual Memory Dump Approaches

There are tools that can get a memory dump of the specified process, such as userdump.exe by Microsoft, pd.exe by T.Klein, pmdump.exe by A.Vidstrom, etc., which use OpenProcess and ReadProcessMemory functions or their low-level analogues like ZwReadProcessMemory, KeStackAttachProcess. The review of these tools is outlined in the following papers. The first drawback of these approaches is their vulnerability to malware manipulation which can hinder expected behavior of these functions, for example by hooking them. The second drawback is that a corresponding dump file does not contain enough information for in-depth memory analysis. Some workarounds to solve these problems are presented further in this article.

2.2 Physical Memory Dump Approaches

2.2.1 Kernel Mode Code

Physical memory dump can be obtained on different levels of execution. There are three popular ways to obtain the dump in kernel mode: ZwOpenSection with ZwMapViewOfSection, MmMapIoSpace and MmMapMemoryDumpMdl.

Based on recently published papers and author's own reverse engineering research the internal mechanisms of some common commercial and free memory dump tools have been studied (see Table 1 for the listing of examined tools).

Table 1 Commercial and Free Memory Dump Tools

Tool's title and version	Author
AccessData FTK Imager v.3.1.2.0 20.	AccessData Group
Belkasoft Live RAM Capturer	Belkasoft
Compiled Memory Analysis Tool (CMAT)	J. Okolica, G.Peterson
DumpIt v2.0.0.20130807 RC1	MoonSols Ltd
Encase Forensic v.7.05	Guidance Software
FastDump v2.0.6.9	HBGary
Memory DD v1.3	ManTech International
Memoryze v3.0.0	Mandiant Corporation
ProDiscover Basic Edition v8.2.0.2	Technology PathWays
Redline v1.11	Mandiant Corporation
Winpmem v1.4.1	The Volatility Foundation

It turns out that all these tools use one or several functions described above. Table 2 presents the results of the survey. Functions that are used in the program are marked with symbols «+» and «–».

Unfortunately, KnTDD toolset by GMG Systems Inc was unable to be obtained, but according to this toolset also uses the same functions.

Memory dump can also be acquired and analyzed remotely, these possibilities are already implemented in commercial products, e.g., Toolset's local agent reads physical memory using the above mentioned functions and then transfers data to the server.

Table 2 Program Tools and Their Functions

Tool's name	Memory Dump Window Functions		
	ZwOpenSection, ZwMapViewOfSection	MmMapIoSpace	MmMapMemoryDumpMdl
AccessData FTK Imager	+	–	–
Belkasoft Live RAM Capturer	–	–	+
CMAT	+	–	–
Dumpit	+	+	+
Encase Forensic	+	–	–
FastDump	+	+	–
Memory DD	+	–	–
Memoryze	+	+	–
ProDiscover	+	–	–
RedLine	+	+	–
Winpmem	+	+	–

Similarly to virtual memory dumping approaches malware can prevent memory acquisition, for example by hooking these functions.

Another method to prevent memory acquisition was described by L.Milkovic, where the author suggested hooking functions which save memory pages to HDD or transfer them and manipulate with buffers content. As a result, final memory dump file will not contain pages with hidden objects including processes, drivers or network ports.

This clearly shows that the existing kernel-mode tools are not resilient to sophisticated malware.

2.2.2 VMX-root Mode Code

Let's focus on low-level approaches for memory dump acquisition. With the help of hardware virtualization technology it becomes possible to execute a code (hypervisor) on a more privileged level (VMX-root mode) than operation system's level. Hypervisors can be used to acquire memory dump. This process is described in the following projects.

Unlike the previously mentioned approaches this one is resilient to the most popular malware tricks which prevent memory dump acquisition. At the same time this method only works on systems, which support hardware virtualization and only in case when a previously loaded hypervisor supports nested virtualization.

One disadvantage of this method is its vulnerability to the "Man-In-The-Middle" attack, because malware hypervisor can load itself sooner than a trusted one. With the help of Shadow Page Tables (AMD) and Extended Page Tables (Intel) malware hypervisor can hide memory areas. As a result, the trusted hypervisor cannot read certain memory pages.

Trusted Execution Technology (TXT) by Intel and Secure Extension Mode (SEM) by AMD provides mechanism for a trusted hypervisor loading by means of Trusted Platform Module (TPM). Unfortunately these technologies are also vulnerable.

This approach can be resilient to "Man-In-The-Middle" attack if a legitimate hypervisor is loaded from BIOS. However this case is only possible in laboratory conditions, because the BIOS hypervisor is highly platform dependent and its adaptation requires additional research that involves difficulties.

2.2.3 System Management Mode Code

System Management Mode (SMM) is more privileged than VMX-root mode. SMM provides power management features and backward compatibility. SMM is partially documented and described in the following papers by K.Zmudzinski, S.Embleton. Opportunities of SMM to acquire memory dump were described in the following papers.

Practical applicability of this method is hindered by installing of SMM dispatcher in general motherboard. Another disadvantage of this approach is the necessity of PC rebooting that is not always possible. This approach can be applied in some older models of motherboards. Adapting this method to new computers requires serious and non-trivial research.

2.2.4 Active Management Technology Code

On computers supporting Active Management Technology (AMT), which is a part of Intel Management Engine (ME), another memory acquisition method can be implemented. AMT code is executed in additional process unit which is located either in the Northbridge or Southbridge. As a result this code is more privileged than VMX-root mode code or SMM code.

The following papers cover this mode from the information security point of view. Due to the fact that malware can be executed in this mode, we can state that memory dumping can operate in this mode too.

Widespread use of this method in practice is hampered by the lack of comprehensive documentation on AMT and ME.

2.2.5 Hardware Approaches

F. Davies in mentions that with I/O Memory Management Unit technology (IOMMU) by AMD and Virtualization Technology for Directed I/O (VT-d) by Intel software approaches to memory acquisition will show poor performance if compared with hardware approaches. Therefore let us focus on hardware approaches to memory dump.

Capabilities of DMA devices such as PCI (PCIe) were used in the following tools: Tribble PCI Card by B.Carrier and J.Grand, Co-Pilot by Komoku and Microsoft, CaptureGuard PCIeCard by WindowsScope, RAM Capture Tool by BBN Technologies. Capabilities of FireWire bus to acquire RAM memory were described by A.Boileau. The applicability of hardware interfaces USB, eSATA, DisplayPort, Thunderbolt and others for accessing physical memory is described by R.Breuk and A.Spruyt. These devices have a similar structure and are hardware boards, which are connected to a PC and designed for memory forensics.

Standard equipment can also be used to memory dump acquisition. For instance, usage of Graphics Address Remapping Table (GART) is described by N.Lawson, D.Goldsmith amd T.Ptacek. Y.Bulygin designed DeepWatch for memory dump acquisition with the help of the Northbridge integrated controller.

It is essential to point out that malware can prevent memory dump acquisition even by hardware approaches. For example, External Access Protection technology by AMD is able to shadow memory pages from peripherals. J.Rutkowska describes how to hide memory areas from peripheral access by reprogramming the Northbridge controller. Modifications in address dispatch tables in the Northbridge controller can hide physical memory regions.

Despite the fact that hardware approaches are resistant to common ways of hidden malicious software, they are only applicable under laboratory conditions, because of applicability and replication inconvenience.

2.2.6 Other Software Approaches

Among other tools for memory dump acquisition another approach was suggested with emulation tools such as VmWare, Vbox and others. This approach is based on suspending the virtual machine. As a result the virtual machine paging file will contain the required data (*.vmem file in VmWare case). Malware is able to detect such emulation tools and hamper their work.

Memory areas can also be acquired with the help of common operating system tools. Papers by Carvey, Vomel and Freiling, Milkovic, and Okolica describe how to use pagefile.sys, crash dump file, hyberfil.sys for memory dump acquisition.

Page file is used for temporary storage of memory pages. According to papers by Zhao and Ruff the pagefile.sys does not contain full memory dump. To restore its content this file has to be merged with RAM dump, which poses additional difficulties.

Crash dump file (memory.dmp) will be created after a Windows system is crushed. This file contains information concerning the event details which caused the system crash. Microsoft developed a way to generate this file artificially – CrashOnCtrlScroll. The disadvantage is that the crash dump is created only after the system is crashed, which is inconvenient for commodity systems. Crash dump file also has some other disadvantages.

Windows OS family starting with Vista adds support for hibernation mode. It causes creation of a hibernation system file (hyberfil.sys) which contains data about a current state of the system. On the one hand this file includes memory pages, but on the other hand it can hardly be used in deep forensic

analysis. S.Vomel and F.Freiling with reference to Russinovich point out that hyberfil.sys cannot be used to restore full RAM because of the limited quantity and quality of the saved pages file, this drawback is mentioned in.

There are a number of research projects based on the idea of 'cold booting', a method by S.Johannes, C.Michael. Freezing memory chips, their removal from the computer and placing them into another PC to analyze memory content was suggested by Halderman et al. Despite the fact that this idea has been extensively tested by several authors, it is still far from commodity production. This fact undoubtedly can be considered as a drawback.

Another proof-of-concept project is BodySnatcher by Schatz which suggested using alternative OS injection on the top of the existing OS. The main disadvantage of BodySnatcher is its poor usability, other disadvantages are described in the papers by Ruff and Vomel and Freiling.

The latest approach to acquire a physical memory dump was offered by Stuttgen and Cohen in 'Anti-Forensic Resilient Memory Acquisition'. With the help of rewriting page frame number in page table entries they got access to the required physical page. Their approach is resilient to modern anti-forensic techniques like hooking, but it is rather slow and vulnerable to rootkits which directly manipulate kernel pages table.

2.3 Conclusion

The analysis shows that the existing approaches and tools of memory dump acquisition do not fully comply with the current requirements:

1. Approaches based on Windows OS functions are vulnerable to intruder's attacks. VMX, SMM, AMT and hardware methods are difficult to use in industrial environments. They are more suitable for a specialized laboratory with highly qualified experts. Other research projects approaches are difficult to apply in practice.
2. Due to the fact that some memory pages are stored in a paging file, RAM dump does not contain complete data. This is especially obvious for PCs with low RAM.
3. The raw physical memory dump is not suitable for extracting useful information because relationships between the virtual and physical address spaces are lost. To overcome this fact additional work has to be done, for example lookup of EPROCESS structures by Burdach or KPCR structures by Zwang, Wang. This work involves a lot of difficulties.

It is essential to develop new detection software, which is resilient to common rootkits tricks. This software should pose great opportunities for memory dump analysis and forensics usage.

3. NEW MEMORY DUMP APPROACH

3.1 Overview

Researchers generally analyze the memory of the target process or kernel mode memory. In this paper we will be focused on the process context. We can use its copy from Windows integrated process such as notepad.exe or run an additionally installed process. As a result we can do research of either user mode or kernel mode memory.

To acquire memory dump for the specified process we run it and then attach to its context. To achieve anti-hook protection we use own low-level analogues of the following functions ZwCreateProcess and KeAttachProcess. As a result malware hooks are unable to hamper the memory acquisition.

Our analysis system does not only allow us to search different binary and text templates, but also do in-deep memory analysis. An example of such analysis will be given later.

The proposed system includes various tools to solve a lot of different memory content analysis tasks for the target program. It helps to investigate and detect malware and rootkits, reverse-engineer processes, conduct forensic research etc.

3.2 Details

3.2.1 Basics

It is suggested to launch one of the common processes or choose an already running one to analyze kernel mode memory. One of the possible scenarios may be the following: run notepad.exe, attach to it and dump memory, detach from it and terminate. When low level protected analogues of functions ZwCreateProcess and KeAttachProcess were developed, they were based on Windows NT4.0 source code, Windows Research Kernel Source Code, and ReactOS source code.

As a result of memory dumping two files will be created: the first file with memory pages 'dump.log' and the second one 'struct.log' with information about page virtual addresses and their offsets in 'dump.log'. Additional information about structures addresses, which are necessary for analysis, for example, EPROCESS list, KDBG, KPCR and etc. are saved into separate files. Examples of these files for analysis will be discussed later.

During dumping the content of each valid memory page is saved into 'dump.log' after buffering. Additional data is saved into 'struct.log', which includes virtual addresses of the pages beginning and end, offsets in 'dump.log' up to the beginning of the copied page. With the help of 'struct.log' and 'dump.log' it is possible to read page content, which corresponds to known virtual addresses and vice versa. Handling of these file is described in Section 0.

3.2.2 Memory Dump Approach

Figure 1 shows an example of saving page #3 in 32-bit mode. It is well known that each virtual memory page corresponds to a page or frame in RAM. The corresponding pages sizes match but their order is usually different.
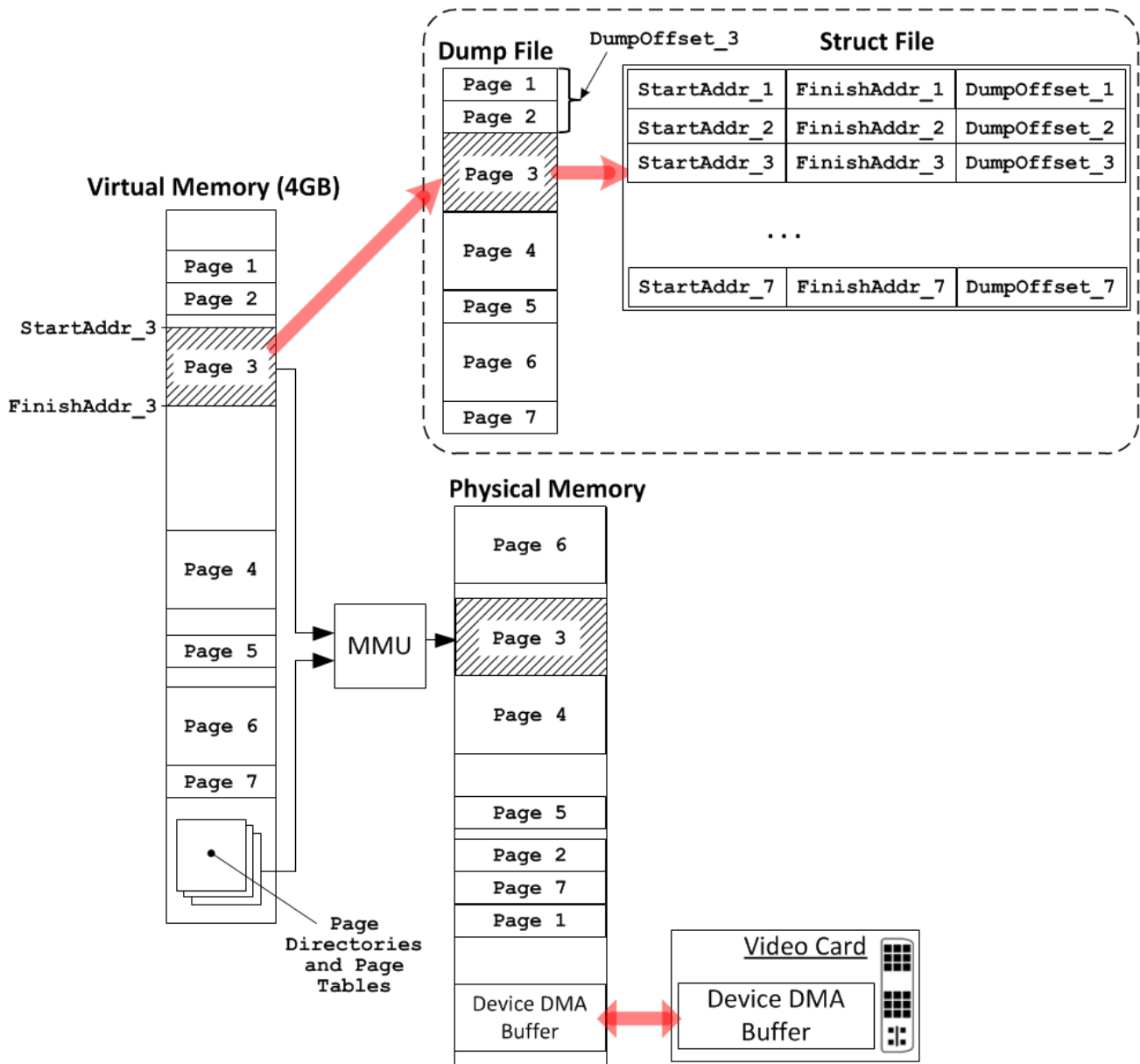
Figure 1 Memory Dump Acquisition Process

Memory dump is acquired after walking through system tables such as Page Directories, Page Tables and others. Details of this walk depend on paging mode, whether Physical Address Extension (PAE) is enabled or not and also on 32-bit or 64-bit Windows versions.

Algorithms of walking for 32-bit Windows OS are similar whether PAE is enabled or not. In case of 64-bit the walking algorithm is similar but additional tables have to be taken into consideration. Therefore let us focus on the algorithm of walking for 32-bit Windows OS.

Memory dumping algorithm is based on the following tables walking workflow:

1. Walk successively through the Page Directory entries. Check the P flag of each entry.
2. If PDE.P is 0, go to the next entry. If PDE.P is 1, check the PS flag.
3. If PDE.PS is 1, save the corresponding memory page. Its size depends on whether PAE is enabled or not and is equal to 2-MByte or 4-MByte correspondingly.

4. If PDE.PS is 0, the current entry corresponds to Page Table, which contains information about with 4-KByte page. Go to this Page Table.
5. In a similar way walk successively through Page Table and check P flag of entry. If PTE.P is 0, go to the next entry, otherwise save the corresponding 4-KByte memory page.

Saving of each page is performed with buffering instead of getting copied directly into the file as it is done in a number of other tools. When the buffer is full, its content is being archived and encrypted and after that the results are saved into the 'dump.log'. Buffering helps to prevent these pages from modifying and increases the overall program performance.

Main features of the memory dump approach:

- The walk through the pages tables has to be done from high addresses up to low ones to exclude loading of empty pages. While walking from the first to the last entry CPU loads a lot of empty pages. Walking has to be started from the last entry to avoid this.
- The walk has to be implemented at PASSIVE_LEVEL IRQL, because only at this level accessing a page which is swapped to HDD means that its content is automatically loaded into memory.
- When we access a memory page related to device direct memory access (DMA) buffer system crash occurs in Windows Vista, 7 and 8. These critical exceptions cannot be caught by try and except. To prevent the crash these memory pages have to be ignored, see the details below.

Technique of ignoring memory pages of DMA devices

According to specification for PCIe (PCI) devices (for example modern network devices, video cards and others) they are able to directly access RAM. While we walk through virtual addresses OS functions allow getting physical addresses ranges of devices. To deal with this it is necessary to use Page Frame Number (PFN), which is a part of Page Table or Page Directory entries. The corresponding physical address is defined in the following way: PFN*0x1000. On the other hand virtual memory page address is determined with the help of indexes in Page Table and Page Directory.

To check whether this virtual memory page corresponds to the pages of DMA devices, the following steps have to be performed:

1. With the help of library functions exported from Setupapi.lib and Cfgmgr32.lib get the ranges of physical addresses which correspond to PCI devices ('prohibited list').
2. While walking through Page Table and Page Directory check each entry whether corresponding physical address belongs to 'prohibited list'. Once it does, skip this entry and check another one according to the algorithm.
3. If it does not, save the corresponding page according to the algorithm.

This technique has been successfully tested on several computers with different hardware and equipment. Access to the following PCI devices buffers (see Table 3) caused a system crash as described above.

Table 3 PCI Devices Which Caused a System Crash

PC and OS	Devices which cause a system crash
Asus P5Q, Win7 32	• NVIDIA GeForce GT 520; • Atheros AR8121/AR8113/AR8114 PCI-E Ethernet Controller, integrated into motherboard Asus P5Q.
Z800, Win7 32	• NVIDIA Quadro FX 580; • D-Link DGE-560SX Single Fiber Gigabit Ethernet PCI-E Adapter (rev.A), additional plug-in device.
Z600, Win7 32	
Shuttle XS36V, Win7 32	No problem in basic configuration.

The disadvantage of this method lies in ignoring physical memory ranges of all PCI devices to avoid crashes, no matter whether DMA is supported and used by this device or not. However it is possible to manually set the physical memory ranges that should be ignored. This disadvantage does not diminish the importance of MASHKA, because the essential structures such as EPROCESS and DRIVER_OBJECT cannot be located in the memory of these devices.

3.2.3 The Acquired Data Processing

Once 'dump.log', 'struct.log' and other files are received, they are processed either locally on a current PC or remotely after transferring these files to the remote host.

The main task of the dump analysis is gaining access to the dump data content located on the required virtual address. This operation is hampered in the existing products because there is not enough information about paging: whether virtual addresses correspond to physical addresses.

To achieve the correspondence between virtual addresses in original memory and offsets values in memory dump file we need additional two files−'dump.log' and 'struct.log' simultaneously.

We will use the following abbreviations 'ODUF', 'VALF' and 'VAOM'. 'VALF' means the virtual addresses of the loaded memory dump file, 'ODUF' means corresponding offsets in dump file. File 'struct.log' contains virtual memory ranges of 'VAOM' and corresponding dump file offsets 'ODUF'. 'VAOM' is virtual address of the original memory; its values are used for further search for the structures, which contain the required virtual address.

Making memory dump analysis it is often necessary to use 'dump.log' and 'struct.log' files simultaneously and convert 'ODUF', 'VALF' and 'VAOM' into each other.

Let us look at this process.

1) 'VAOM' -> 'ODUF'
   As a result of the lookup in the 'struct.log' file we find i-entry, which contains virtual memory ranges, so that target value of 'VAOM' belongs to its range. 'ODUF' is defined in the following way:
   $ODUF = DumpOffset[i] + (FinishAddr[i] - VAOM)$.
2) 'ODUF' -> 'VAOM'
   As a result of the lookup in the 'struct.log' file we find i-entry, so that $Offset[i] <= ODUF < Offset[i+1]$, where $Offset[i+1]$ means Offset of the following (i+1)-entry. 'VAOM' is defined in the following way:
   $VAOM = FinishAddr[i] + (ODUF - Offset[i])$.
3) 'ODUF' <-> 'VALF' and 'VAOM' <-> 'VALF'

Values of 'ODUF' and 'VALF' are different by the value of starting address of the loaded dump file: VALF = ODUF + LoadAddr and vice versa. Having this equation it is possible to convert 'VAOM' <-> 'VALF'

These operations facilitate the in-depth analysis of the dump. Examples will be given below.

3.3 How to use MASHKA in Memory Forensics Tasks

Memory analysis basic operations include text or binary signatures lookups through a memory dump. Current version of MASHKA can do multi-threaded lookup for the following objects: one byte (char) or wide-character (wchar_t) strings and byte fragments, for example addresses values.

When an object has been found, its VAOM, VALF and ODUF are forwarded for further research. The lookup is conducted from the beginning of the 'dump.log', byte-by-byte or in 4 byte order for special structures such as EPROCESS and DRIVER_OBJECT.

It is also possible to search the addresses, whose values are around the target VAOM address. For example, some system structures store information about the string values in the form of UNICODE_STRING or PUNICODE_STRING. To research and detect these structures it is necessary to conduct a search for the target wchar- string, and then further search for each discovered address of VAOM string. In case of PUNICODE_STRING it is necessary to conduct a search for the (VAOM-4) value, where 4 is 'Buffer' field offset from the beginning of UNICODE_STRING.

It is possible to search for byte fragment of target file header or one of its sections.

It is possible to carry out the following operations on the information obtained: walking through singly and doubly linked lists of structures and getting detailed information for further analysis, and also coping data located in the target virtual address range.

As an example, the following iterative research workflow for driver detection with the help of memory dump and WinDbg can be given:

1. Load OS Windows in debug mode under WinDbg control.
2. Install a driver with the specified 'ServiceName' and 'DisplayName' located in 'BinaryPathName'. Run this operation on the specified machine with the help of System Control Manager (SCM).
3. Hide this driver by well-known technique, based on PsLoadedModuleList.
4. Check the system with the help of some popular anti rootkit tools. This tool has to detect a deliberately hidden driver.
5. Get memory dump with the help of MASHKA. Copy 'dump.log', 'struct.log' and other essential files to the host machine.
6. Search for one byte and wide-character string containing, 'ServiceName', 'DisplayName' and 'BinaryPathName'. Save the received 'VAOM'.
7. Freeze the target machine with the help of WinDbg.
8. With the help of WinDbg change VAOM string content values. For example, patch one character 'A'-'Z' to the beginning of each string.
9. Check the system with the help of anti-rootkit for the second time. As a result the detection tools will give a changed name. Knowing the corresponding string names and 'VAOM' run further analysis. Sort out the details in corresponding data lists, as well as detection mechanism of the anti-rootkit.

4. NEW ROOTKITS DETECTION TOOL

This section is focused on the analysis of the existing approaches to hidden objects (processes and drivers) detection. Their drawbacks will be pointed out and author's detection approaches will be suggested, which uses Dynamic Bit Signature (DBS) for processes and Rating Point Inspection (RPI)

for drivers. Finally, we will describe some currently known disadvantages of the approaches and ways to overcome them and for improvements.

4.1 Problem Statement

Cybercrime has become more and more sophisticated. Recently there has been a clear tendency or shift in computer attacks from mass infections to targeted attacks. E.Kaspersky assessed 'IT threats that have evolved from cyber hooliganism, via cybercrime to cyber warfare'. The new type of malware appeared such as Stuxnet, Duqu, Flamer, Gauss, that many antivirus companies call a cyber-weapon. Another example is spy network 'Red October' that stole large amounts of data from diplomatic, government and science agencies in Europe, Middle East and Central Asia for 5 years. Sophisticated intruder protection and heuristics did not prevent malware infection and subsequent activity.

Malware developers are working on long term attacks, which will give hackers an ongoing and virtually undetectable access to the target system. To ensure that malware has to use special rootkit mechanisms, which provide hiding of the following OS objects: processes, threads, drivers or services.

According to J.Rutkowska there are two types of rootkit mechanisms to hide objects from built-in tools (for example 'taskmgr.exe' to get the processes list) which work in OS: functions-hooking mechanisms and direct kernel object manipulations (DKOM). Hooking is relatively simple to detect and will not be examined in this paper. Yet DKOM implementation uses minimal number of changes, which makes it the most complicated case for detection. This case will be discussed later.

Current anti-rootkit approaches have significant disadvantages, i.e. they are either vulnerable, or their portability implies serious research.

Therefore the goal is to develop a new detection approach which is resilient to common rootkits tricks. Let us analyze how current detection tools work.

After the OS object has been executed operation system creates a structure which solely corresponds to this OS object, see Figure 2.
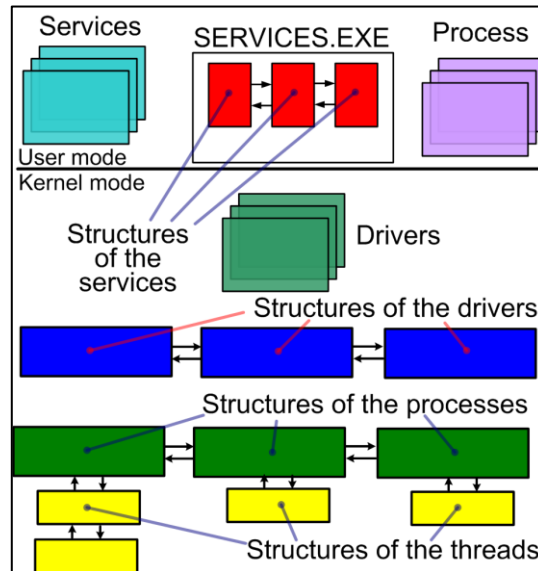

Figure 2 Processes, Threads, Drivers and Services Structures in Windows OS

Structures of different types are labeled as rectangles, for example, the process structures (EPROCESS) are labeled green. Object structures join OS doubly linked lists (see arrows on the above Figure 2). From these lists built-in tools get object information for object management.

The idea of DKOM suggested by G.Hoglund was in concealing an OS object through the unlinking the corresponding structure. This does not crash the OS or yields to object termination, but built-in tools cannot further detect objects.

Next chapters describe existing methods and tools to detect hidden objects and their analysis.

4.2 Analysis of Current Approaches to Rootkit Detection in Face of Oppositions

One of the most popular ways to detect rootkits at runtime is known as cross-view detection, which relies on the fact that there are several ways to collect the same information about OS objects. Cross-view detection typically utilizes both high-level and low-level mechanisms to collect information. The high-level mechanism is based on standard system functions to enumerate OS objects. The low-level mechanisms are based on data from some heuristic analyzers, additional object structure lists, signature scans and other heuristic.

We will analyze existing approaches according to a number of criteria, such as resilience to common rootkit tricks, portability to new versions of Windows and others.

Heuristic analyzer tracks programs activity, analyzes the collected data and blocks the program if its behavior is similar to a malicious one. The main disadvantage of this approach is that it blocks the program only after a certain amount of its activity has been collected during tracking. Another disadvantage is its vulnerability to rootkit countermeasures. Also heuristic analyzer must be started before malware, which is not always possible.

Information about running objects is often duplicated in different systems' lists. It is possible to use this data for objects detection. Is this case hidden object detection is based on data comparison obtained from various lists. This method was implemented in Tuluka Kernel Inspector, TDSS killer by Kaspersky lab and others. To hijack this detection the malware is able to modify all the needed lists to hide its own presence. As a result malware activity will not be detected.

Signature scan is based on byte to byte search of fragments of objects structures in memory. This method has been implemented in GMER, PowerTool, XueTr and others. It is important to point out that structure sizes and their content change in new Windows versions (after some updates, service packs) as for EPROCESS structure. To deal with that, this method needs adaptation, which is often difficult because it requires manual adjustments.

It is possible to prevent hidden object detection by signature scan. To achieve this malware may modify some structure values, which are used by signature scan. These modifications cannot crash the system or stop malware activity but make signature scan useless. One reason for this is that the decision is based only on the signature coincidence for the whole structure. If at least one byte does not match, the signature scan will miss the structure.

A similar method to prevent hidden object detection was proposed by T.Haruyama, H.Suzuki in 'One-byte Modification for Breaking Memory Forensic Analysis'. The prevention is based on modification of systems' structures values, which caused the situation when the detection tools were disabled.

Let us analyze the mentioned approaches with regard to processes structures (EPROCESS) and drivers structures (DRIVER_OBJECT) because they are often used in malware attacks.

4.2.1 Inside EPROCESS Detection

When a process has been started a new content is created, and information about new object is added to different systems lists. A significant number of such lists make it difficult to hide the process well; therefore we usually speak about hiding the process only from built-in tools. There are a lot of approaches to process detection, so let us name some of them. There are some approaches based on additional objects structures lists, such as processes list from CSRSS.EXE, thread-based scheduling list and others. There are some heuristic analyzer approaches which are based on hooking functions,

such as SwapContext or KiFastCallEntry. The Volatility Project includes various plugins list to stealth process detection.

Grizzard's approach was based on locating x86 paging structures in memory images. Another MAS tool which was described in his paper uses memory crash dump file to rootkit detection, for this reason it is impossible to apply this method in commodity systems.

Another process detection approach has been suggested by Schuster. This approach is based on the fact that values of some EPROCESS fields are either known or exceed the constant, for example 0x8000_0000. Author's approach has a number of important disadvantages: it is difficult to achieve its portability on different versions of Windows OS, as well as it is vulnerable to field modifications.

Another approach was based on signature search. The authors suggest new graphs signatures, which can evaluate contingent structures in Linux OS. This method is also vulnerable to specific byte modifications. It is also difficult to make and test these graphs signatures for new Windows versions, because it requires a specialist's involvement.

Schuster's approach was presented in the paper 'Robust Signatures for Kernel Data Structures'. It proposed including only robust fields in EPROCESS signature. If malware modifies one of these fields, the system crashes. To search these robust fields the author suggested control memory access with the help of adapted XEN hypervisor and VMware. The major drawback of this approach is its applicability only to structures with a lot of elements like EPROCESS, for which it is possible to find robust signature. Therefore it is impossible to apply this method to DRIVER_OBJECT structure detection.

4.2.2 Inside DRIVER_OBJECT Detection

In comparison with process creation, driver loading causes much fewer system modifications, which makes it possible to achieve better drivers hiding.

Drivers hiding was described in popular books such as 'Rootkits: Subverting the Windows Kernel' by G.Hoglund and J.Butler, and in new B. Blunden's book 'The Rootkit Arsenal: Escape and Evasion in the Dark Corners of the System'. It is necessary to mention drivers lists, which are not used by built-in tools: PsLoadModuleList, ObjectDirectory lists, Service Control Manager (SCM) drivers list.

Detection of hidden drivers is very similar to stealth process detection.

Schuster's signature approach has been adapted by W.Tsaur and L.Yeh to drivers detection. However, their approach is also vulnerable to target byte modification.

The following non-built-in well-known tools which support Windows 8 are: XueTr by linxer, PowerTool by ithurricane, TDSSKiller by Kaspersky Lab. In terms of driver detection three first tools have similar detection algorithms, which are based on byte-to-byte signature search among DRIVER_OBJECT structures. TDSSKiller uses a completely different detection algorithm. Its algorithm uses a system list, that holds information about new drivers added by SCM. By field values modifications it is possible to hide specified driver structures from all these tools. There are modifications that do not stop drivers or corrupt OS functionality.

It will be discussed further how to improve Schuster's idea to create a rootkit detection approach, which is both resilient to byte modification and still portable to new Windows versions for both 32-bit and 64-bit editions.

4.3 New Stealth Processes and Drivers Detection Approach

4.3.1 Dynamic Bit Signature (DBS) for EPROCESS Detection

Let us look first at the dynamic byte signature approach and how to apply it to process detection. After that we will describe dynamic bit signature approach and present its advantages.

To detect process structures, hidden with the help of DKOM method, we need to analyze the content of EPROCESS structures. Our goal is to find some common peculiarities between EPROCESS structures of different processes. Different bytes are illustrated on Figure 3 as squares with different colors. The corresponding squares have identical colors if byte values are the same.
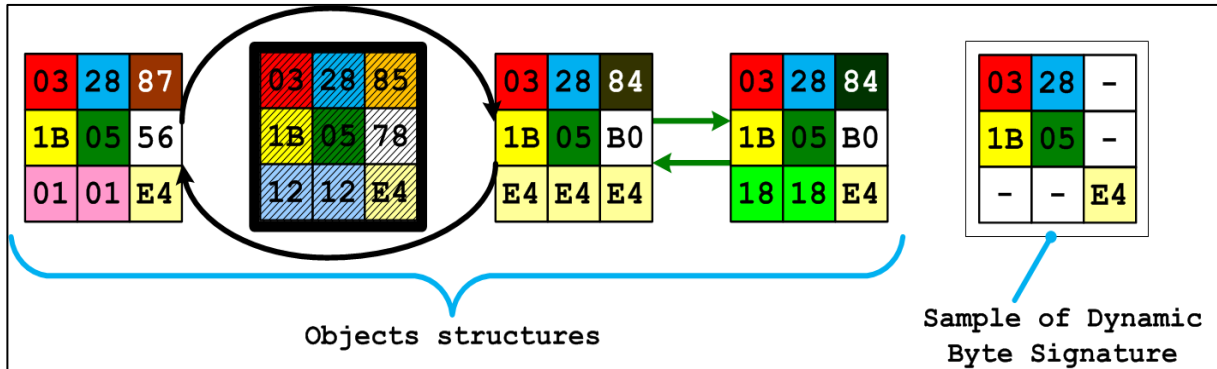

Figure 3 Objects Structures Typical Design

It is obvious that initial bytes of each structure are identical, but further bytes are different. The conclusion was done, that if we search for some typical EPROCESS structure fragments it is possible to find all EPROCESS structures regardless of whether they are hidden or not. It is shown below how to do this.

Stealth process detection approach:

1. Create dynamic bit signature (DBS) as a template, which matches to all EPROCESS.
2. With the help of probabilistic search of DBS in kernel memory find all EPROCESS structures, either hidden or not. As a result, get the author's list.
3. Compare the author's list with a list of processes obtained by standard means of the OS, e.g. NtQuerySystemInformation.

Dynamic bytes signature includes only the bytes, whose values are the same for all EPROCESS structures, which are in the list. For example, all EPROCESS structures contain the same byte in their center. It is labeled on Figure 3 as a green square ('05').

This byte is automatically added to the signature.

This signature in used to search EPROCESS structures manually. This is done with the help of byte-to-byte search in kernel memory. For each memory fragment the number of matches with DBS-signature is calculated. If for the current memory fragment the inequality $(\Sigma - \Delta) \leq i \leq \Sigma$ is true, it is considered that the structure of the similar object is found, Σ – is the number of bytes in a signature, Δ – threshold value (for example Δ may be equal to $(\Sigma * 0.8)$), i – is the number of matches for the current memory fragment with DBS-signature. If for some memory region this inequality is false, we skip this region and continue analysis with the next memory fragments until all the memory is analyzed.

As a result the full processes structures list based on DBS-signature matching will be obtained.

The conclusion if hidden processes are present is made after comparing DBS-matching list with the list obtained by NtQuerySystemInformation. This approach has been successfully tested for both cases of deliberately hidden objects and for real rootkits, such as Virus.Win32.Sality.q (Kaspersky Lab) and Trojan.Win32.VB.aqt (Kaspersky Lab).

It is important to emphasize that EPROCESS structure includes a lot of fields, whose values are linked with other kernel mode structures. Therefore these values exceed the values of 0x8000_0000. This fact is partly used in the Schuster's paper, but his approach is still vulnerable to byte modification and

needs EPROCESS signature update when the new Windows version is released. We propose to improve the bytes-based signature approach with a bits-based one, which works in the similar way but on the bits values level.

Such approach has the following advantages:

- By the automatically generated bit-based signature, it is possible to adapt byte-based approach for new Windows versions and SP;
- Due to probabilistic nature of lookups it is possible to find EPROCESS structures even if they were deliberately modified and only 70-80% of data matches the signature. Threshold value can be adjusted manually.

This approach can be used to detect all objects in memory, which have a typical structure, but only if the structure definition is large enough. This method works badly for compact structures, because the amount of false detected structures increases. For DRIVER_OBJECT structures detection, whose sizes are more than 10 times smaller than EPROCESS structure sizes, the proposed approach needs improvements that are described further.

4.3.2 Rating Point Inspection (RPI) for DRIVER_OBJECT Detection

Rating Point Inspection (RPI) is the development of DBS detection approach. The first difference is that we need to manually adjust RPI to specific structure types (such as DRIVER_OBJECT or DEVICE _OBJECT structures). The second difference in case on RPI is the utilization of additional weight matrix for precise matching accounting. We calculate total matching points (score) but not the individual matches themselves. For example, if one of the checks is true, 1, 2, or 3, etc. points are added to the final score. In DBS case we simply summarize the numbers of matches or add only 1 point to the final sum, if the check is true.

The conclusion for DRIVER_OBJECT structure matching is made in the similar way by comparing the score with the threshold value. The threshold value is determined by calculating the same metrics for "not hidden" DRIVER_OBJECT structures, which are located in DirectoryObject.

First let us briefly describe the DRIVER_OBJECT detection technique and then give an explanation:

1. Get memory dump ('dump.log' and 'struct.log'), save the DRIVER_OBJECT structures addresses in 'drvobj.log' file. To do the latter, use ZwOpenDirectoryObject function.
2. Determine 'min_major_function' value.
3. Determine 'global_scope' value.
4. Determine 'global_scope_deep' value.

The following steps (5, 6, 7) are done iteratively, and will be explained further.

5. Perform a byte-to-byte DRIVER_OBJECT structure search with the help of 'is_integrated_driver' function, which calculates the numbers of matching points for each memory region.
6. The conclusion that DRIVER_OBJECT structure is found is made after comparing these matching points from step 5 with the 'global_scope' value, which was obtained on step 3. If this value is not smaller than 'global_scope' value, the DRIVER_OBJECT structure is present. Otherwise calculate the numbers of deep matching points for this memory area with the help of 'is_integrated_driver_deep'. If the structure has been found, go to step 5 and continue lookups.
7. The conclusion that DRIVER_OBJECT structure is found is made after comparing the deep matching points obtained in step 6 with the 'global_scope_deep' value, which was obtained on step 4. If this value is not smaller than 'global_scope_deep' value, the DRIVER_OBJECT structure is present. Otherwise go to step 5 and continue lookups.

8. Repeat steps 6-8 for the whole memory area. As a result, get the RPI-matching list of DRIVER_OBJECT structures.
9. Compare the RPI-matching list with the drivers list, which has been obtained on step 1.

Further, steps 2, 3, and 4 will be described in details further. Steps 6 and 7 are 'if-else' statements.

Details of step 2. Determine 'min_major_function' value. Use ZwOpenDirectoryObject function to obtain the list of DRIVER_OBJECT structures. For each DRIVER_OBJECT structure calculate the maximum number of functions' addresses from MajorFunction, whose addresses are the same, with the help of 'max_same_major_functions'. From these values select the minimum – 'min_major_function'.

Details of step 3. Determine 'global_scope' value. Use ZwOpenDirectoryObject function to obtain the list of DRIVER_OBJECT structures. For each DRIVER_OBJECT structure calculate the numbers of points with the help of Table 4. If one of the conditions is false, we add 0 points to the total number of matching points. Total matching score is calculated as a result of checking all the conditions in the table. For example, if all the conditions are true, apart from the second, the total score is 10. Among these values select the minimum – 'global_scope'.

Table 4 Weight Matrix to Calculate 'global_scope'

Condition	Score
if (DRIVER_OBJECT_32.Type == 0x04)	2
if (DRIVER_OBJECT_32.Size == 0xa8)	4
if (chk_unicode_string(&DRIVER_OBJECT_32.DriverName))	2
if (chk_unicode_string(DRIVER_OBJECT_32.HardwareDatabase))	2
if ((DRIVER_OBJECT_32.MajorFunction[0]) >> 31)	2
if (max_same_major_functions(&DRIVER_OBJECT_32) >= min_major_function)	2

Function '**chk_unicode_string**' checks whether the UNICODE_STRING structure is valid. This is done by checking conditions from the Table 5. Construction 'iswprint(UNICODE_STRING)' specifies checking of all the characters of the corresponding buffer using a 'iswprint' function.

Table 5 The 'chk_unicode_string' Function

Condition	Result
(UNICODE_STRING.MaximumLength >= UNICODE_STRING.Length) && (UNICODE_STRING.Buffer!=NULL) && iswprint(UNICODE_STRING)	true or false

Details of step 4. Determine 'global_scope_deep' value. Use ZwOpenDirectoryObject function to obtain the list of DRIVER_OBJECT structures. For each DRIVER_OBJECT structure with the help of Table 6 calculate the numbers of matching points. Among these values select the minimum 'global_scope_deep'.

Table 6 Weight Matrix to Calculate 'global_scope_deep'

Condition	Score
if (DRIVER_OBJECT_32.Type == 0x04)	2
if (DRIVER_OBJECT_32.Size == 0xa8)	2
if (DRIVER_OBJECT_32.DriverStart >> 31)	2
if (DRIVER_OBJECT_32.DriverStart % 0x1000 == 0)	2
if (DRIVER_OBJECT_32.DriverSize % 0x1000 == 0)	2
if (check_function_prologue(DRIVER_OBJECT_32.DriverStart))	4
if (DRIVER_OBJECT_32.DriverExtension >> 31)	2
K = chk_unicode_string2(&DRIVER_OBJECT_32.DriverName)	K
chk_unicode_string(DRIVER_OBJECT_32.HardwareDatabase)	2
if ((DRIVER_OBJECT_32.MajorFunction[0]) >> 31)	2
if (max_same_major_functions(&DRIVER_OBJECT_32) >= min_major_function)	2

The function 'check_function_prologue' checks whether the conditions from the Table 7 are true. This check is repeated for first 16 memory bytes of each memory region (for (int i = 0; i < 0x10 ; i++)).

Table 7 The 'check_function_prologue(addr)' Function

Condition	Result
If (((addr[i+0] == 0x55) && (addr[i+1] == 0x89) && (addr[i+2] == 0xe5)) || ((addr[i+0] == 0x55) && (addr[i+1] == 0x8b) && (addr[i+2] == 0xec)) || ((addr[i+0] == 0x53) && (addr[i+1] == 0x56)) || ((addr[i+0] == 0x56) && (addr[i+1] == 0x57)) || ((addr[i+0] == 0x56) && (addr[i+1] == 0x57)) || ((addr[i+0] == 0x8b) && (addr[i+1] == 0xff)))	true or false

Function 'chk_unicode_string2' is determined in Table 8.

Table 8 The 'chk_unicode_string2(PUNICODE_STRING pDriverName)' Function

Condition	Score
if (pDriverName->MaximumLength >= pDriverName->Length)	2
if ((pDriverName->MaximumLength <= 0x50) && (pDriverName->Length <= 0x50))	4
if (chk_unicode_string(pDriverName))	2
if (_memicmp(pDriverName->Buffer, L".sys", pDriverName->MaximumLength))	2
if (wcslen(pDriverName->Buffer) <= pDriverName->Length)	2

RPI features and its further development:

It is possible to improve the function 'check_function_prologue' by adding an intelligent analyzer, which will detect modified function prologue. It is especially useful when malware employs any kind of armoring (e.g., packers, cryptors).

Also, it is possible for the detected hidden driver to look up its MD5 hash or name through Google search engine. Similar functionality has Process Explorer by M.Russinovich. It is well-known that sections contents on binary file in HDD or that was loaded in memory do not differ much.

The RPI approach has been successfully tested for both cases of deliberately hidden objects, for real rootkits and for hidden drivers, which were loaded with the help of *ATSIV* utility by Linchpin Labs and OSR. In the latter case all existing tools such as PowerTool, TDSSKiller, Xuetr cannot detect a hidden driver, but the proposed method can. YouTube video of these tools with comments is here.

In 'Identifying Rootkit Infections Using a New Windows Hidden-driver-based Rootkit' it was proposed to utilize existing link between DRIVER_OBJECT and DEVICE_OBJECT structures to search for DRIVER_OBJECT structure. Unfortunately this link is optional and even conventional drivers structures may not have this relationship. It makes no sense to check this link. However the RPI approach can be complemented by inspections of such links.

5. DISCUSSION AND FUTURE WORK

The presented MASHKA system has a number of advantages:

- Memory dump and analysis system, which is based on two shared files, have good opportunities for in-depth memory analysis and allow to find the hidden objects–processes and drivers. The first file contains pages contents and the second file contains corresponding sets of matches between virtual addresses and pages offsets.
- Protected implementation of memory dump avoids disruption from popular rootkits tricks.
- Bit-based signature approach provides the most profound inspection of system structures without manual work.
- Dynamic signature makes it possible to generate templates for byte-to-byte lookup or define signatures without a detailed study of the structure definition.
- Due to the fact that the matching conclusion is made with even partial matching to the signature, it is possible to detect even deliberately modified objects structures, where tools based on the idea of exact matching with the signature will miss the modified structure (e.g., Schuster's approach [89], GMER toolkit).

It is important to discuss how to use MASHKA to research and detect rootkits, which use modification of the page fault handler to hide memory pages, so called 'Shadow Walker'-like Rootkits. The bottleneck in MASHKA is linear search of structures templates, it is impossible to use GPU to increase its productivity. Logical development of this system is partial transition to the cloud – Anti Rootkit as a Service. The fact that vast majority of kernel mode structures are loaded into memory closely to each other was revealed. With the help of this fact it is possible to improve rootkit detection method. The cases of MASHKA application and implementation in education will be described later.

5.1 Detection Shadow Walker-like Rootkits

It is important to describe Shadow Walker rootkit (SW), which was presented by S.Sparks and J.Butler at the Black Hat conference in 2006. Despite the time passed this approach is still relevant. This rootkit can hide memory areas with the help of hooking the page fault interrupt handler. As a result, when accessing the memory pages containing the rootkit, their contents are replaced with false values.

Existing popular software does not detect rootkits of this type. Some authors propose to detect the rootkit using either program code, which works in more privileged mode than operation system (e.g. VMX mode or SMM), or hardware memory dump tools.

According to WindowsSCOPE this rootkit can be detected with the help of Interrupt Descriptor Table (IDT) analysis, because if SW has been installed, the page fault (#PF) handler is modified.

It is possible to detect this type of rootkits with MASHKA too. During the memory page walk we need to measure the duration of the memory page access. We need to make two successive attempts to access memory page. During the first access the memory page data loading occurs from page file to memory and system buffers (such as TLB) initialization occurs. The second memory access occurs when measuring the duration of memory page access. The memory region with too large access duration is the stealth memory region. Gaining access to the contents of this region depends on the rootkit implementation. For example it is possible to modify #PF handler. As a result, it is possible to control memory access and read hidden memory regions.

5.2 GPU Utilization in Memory Forensics

Detection of hidden objects occurs by memory lookups. Current version of MASHKA is based on C++ binary code with 'OpenMP' technology, which is provided by Microsoft Visual C++ compiler. However, the observed detection time can be significantly improved by utilizing Graphic Processing Unit (GPU) (which is also occasionally called visual processing unit (VPU)) hardware. To do this we need to transfer the dump files to the device memory and perform all the algorithms on the GPU. The algorithms and memory lookups may be easily parallelized so that will speedup the analysis and free CPU resources for common use.

5.3 The Idea of Cloud Anti Rootkit or Anti Rootkit as a Service

It is possible to use MASHKA toolkit system on tablet PC, such as ThinkPad Tablet 2, as well as on PC with low computational capabilities, such as low-cost laptops. The idea of cloud anti-rootkit or anti-rootkit as a service is as follows: data processing will occur remotely, not on the local PC. The separation of memory dumping and analysis processes yields to more reliable and more flexible IT security management infrastructure. More robust and solid dumping process may need very seldom updates but server-side application and algorithms need another maintenance periodicity. SaaS architecture simplifies the administration. The idea of cloud anti-rootkit leads to possibility of toolkit deployment in corporate networks without supplementary access to public Internet or with remote server in the cloud, so authorized users can load their memory dumps into the cloud and get the information whether there is any hidden object or not. While detecting hidden objects the system will provide detailed information and tools to analyze or eliminate these objects depending on usage scenarios.

5.4 The Center of Mass of Kernel Mode Structures

We have discovered another pattern which can be used in detection. Our research revealed that the placement of kernel mode structures such as EPROCESS, DRIVER_OBJECT and located closely to each other in memory. This fact can be used for detection of kernel mode structures. Based on the addresses of DRIVER_OBJECT structures the so-called 'center of mass' of DRIVER_OBJECT data can be found. The 'center of mass' will be located near most of the structures. When checking another memory area we need to assess how close it is to the 'centers of mass'. An additional criterion for detection is nearest to the 'center of mass' of the structure: the probability that the object found is the true structure increases as it approaches the 'center of mass'. We can calculate the 'center of mass' value with the help of addresses of kernel mode structures, which were already loaded in memory as a mean value.

This feature is valid for drivers loaded with the help of built-in mechanism, such as SCM. However, loaded by *ATSIV* utility by Linchpin Labs this peculiarity is disrupted. To make it clear it is proposed to visualize a memory dump, reflecting the structures found. These issues are not covered in this paper.

5.5 Digital Forensics in Education

The proposed system can help students and postgraduate students in Computer Forensics to acquire practical skills in Computer Science. Students can get acquainted with the basics of memory forensics, Windows architecture, examine the program code and memory; investigate the relationships between binary modules loaded into memory. They will be able to learn the structure of user mode and kernel mode memories. The study of system services used to detect hidden objects during the training course may expect from the students to research the process SERVICES.EXE etc. Memory dump process evaluation makes it possible to study and get descriptions of undocumented structures of services that can be further used to search for hidden objects.

As a result, students consolidate their theoretical knowledge about the operating system, its components and their interaction with memory, as well as acquire research skills to get memory structures, which is crucial for solving practical problems of information security: reverse-engineering research and detection of malware, conducting forensic assessment and evaluation.

ACKNOWLEDGEMENTS

AUTHORS BIOGRAPHIES

	Igor Korkin, Ph.D., is a specialist in information security. He works at Moscow Engineering Physics Institute, training post-graduate students and supervising students. He has been engaged in rootkit technologies for over 6 years and has published more than 10 scientific papers. He was a finalist of the RusCrypto conference in 2011, with "Detection of nested virtual machine monitors" report, winner of "Hackers vs. Forensics" on Forum "Positive Hack Days 2012" in Moscow, Russia. He participated in a number of conferences and seminars. His research interests include rootkits and anti-rootkits technologies; secure operating systems; spyware, backdoors and their detection; hardware virtualization; information leakage channels; memory forensics.
	Ivan Nesterov is an HPC Software specialist, system architect since 2000. His main research areas lie in the domain of high performance computing, parallel programming, distributed and storage systems, database design and applications. He finished the Moscow Institute of Physics and Technology (State University) with an M.Sc. in Applied Mathematics and Physics. Software design experience includes high-performance computing complex with hybrid CPU/GPU architecture for cryptography tasks, distributed visualization complex on heterogeneous computing systems with both non-uniform performance and architecture. Has GAZPROM IT Awards for best software application in 2009 and quality award in 2010.

REFERENCES

AccessData Group. FTK. AccessData. (2014). Retrieved on January 14, 2014 from
http://www.accessdata.com/products/digital-forensics/ftk

Albertinih, A. (2011). PE format's infographics. Retrieved on January 14, 2014 from
https://code.google.com/p/corkami/downloads/detail?name=pe-20110117.pdf

AMD64. (2012). AMD64 architecture programmer's manual, volume 2: System programming. Retrieved on January 14, 2014 from support.amd.com/us/Processor_TechDocs/APM_V2_24593.pdf

Arevalo, J. (2013). Step by step to work with your own memory dumps. *eForensics Magazine*, 36-75.

Athreya, M. (2010). Subverting Linux on-the-fly using hardware virtualization technology. Retrieved on January 14, 2014 from http://arch.ece.gatech.edu/pub/athreya.pdf

Aumaitre, D. (2009). A little journey inside Windows memory. *Journal of Computer Virology and Hacking Techniques*, *5*(2), 105-117. doi:10.1007/s11416-008-0112-2

Belkasoft. (2013). Live RAM Capturer. Retrieved on January 14, 2014 from http://forensic.belkasoft.com/en/ram/download.asp

Blunden, B. (2009). *The Rootkit arsenal: Escape and evasion*. Texas, USA: Jones & Bartlett Learning.

Blunden, B. (2012). The Rootkit arsenal: Escape and evasion in the dark corners of the system. Burlington, MA: Jones & Bartlett Publishers.

Boileau. (2011). A. Hit by a bus: Physical access attacks with Firewire. Retrieved on January 14, 2014 from http://www.security-assessment.com/files/presentations/ab_firewire_rux2k6-final.pdf

Breuk, R., & Spruyt, A. (2012). Integrating DMA attacks in exploitation frameworks. Retrieved on January 14, 2014 from http://www.delaat.net/rp/2011-2012/p14/report.pdf

Bulygin, Y. (2008). Chipset based approach to detect virtualization malware a.k.a. DeepWatch. Retrieved on January 14, 2014 from http://www.hakim.ws/BHUSA08/speakers/Bulygin_Detection_of_Rootkits/bh-us-08 bulygin_Chip_Based_Approach_to_Detect_Rootkits.pdf

Burdach, M. (2006). Finding digital evidence in physical memory. Retrieved on January 14, 2014 from http://www.blackhat.com/presentations/bh-federal-06/BH-Fed-06-Burdach/bh-fed-06-burdach-up.pdf

CaptureGUARD. (2012). Physical memory acquisition hardware by WindowsScope. Retrieved on January 14, 2014 from http://www.windowsscope.com/

Carrier, B., & Grand, J. (2004). A hardware-based memory acquisition procedure for digital investigations. *The International Journal of Digital Forensics & Incident Response*, *1*(1), 50-60. doi:10.1016/j.diin.2003.12.001

Carvey, H. (2009). *Windows Forensic Analysis DVD Toolkit*. Burlington, MA: Syngress Press.

Casey, E. (2005). *Handbook of Digital Forensics and Investigation*. Burlington, MA: Elsevier Academic Press.

Chan, E.M. (2011). A framework for live forensics. (Doctoral dissertation). Retrieved on January 14, 2014 from https://www.ideals.illinois.edu/bitstream/handle/2142/24365/Chan_Ellick.pdf

Cohen, M. (2012). The PMEM memory acquisition suite. Retrieved on January 14, 2014 from http://scudette.blogspot.ru/2012/11/the-pmem-memory-acquisition-suite.html

Cohen, M. (2012). Memory forensics with volatility. Retrieved on January 14, 2014 from http://www.dfrws.org/2012/program.shtml

Cohen, M., Bilby, D., & Caronni, G. (2011). Distributed forensics and incident response in the enterprise. Journal Digital Investigation. *The International Journal of Digital Forensics & Incident Response*, *8*, S101-S110. doi:10.1016/j.diin.2011.05.012

Csk (2012). Intel AMT/ME Meet Intel's hardware backdoor. Retrieved on January 14, 2014 from http://www.uberwall.org/bin/download/download/102/lacon12_intel_amt.pdf

Cui, W., Peinado, M., Xu, Z., & Chan, E. (2012). Tracking Rootkit footprints with a practical memory analysis system. Paper presented at the 21ˢᵗ USENIX Security Symposium, USENIX Association Berkeley, CA, USA, August 2012, 42-57.

Datta, A., Franklin, J., Garg, D., & Kaynar, D. (2009). A logic of secure systems and its application to trusted computing. Paper presented at 30ᵗʰ IEEE Symposium on Security and Privacy (S&P), Berkeley, CA, 17-20 May, 21-236. doi:10.1109/SP.2009.16

David, F., Chan, E., Carlyle J., & Campbell, R. (2008). Cloaker: Hardware supported Rootkit concealment. Paper presented at IEEE Symposium on Security and Privacy, Oakland, California, USA, 18-21 May, 296-310. doi:10.1109/SP.2008.8

Davis, M., Bodmer, S., & LeMasters, (2009). *A Hacking Exposed: Malware & Rootkits Secrets & Solutions*. The McGraw-Hill Companies.

Dolan-Gavitt, B., Srivastava, A., Traynor, P., & Giffin, J. (2009). Robust signatures for kernel data structures. Paper presented at the ACM Conference on Computer and Communications Security, Chicago, Illinois, USA, 9-13 November, 1-12.

Dykstra, J., & Sherman, A. (2012). Acquiring forensic evidence from infrastructure-as-a-service cloud computing: Exploring and evaluating tools, trust, and techniques. Retrieved on January 14, 2014 from http://www.csee.umbc.edu/~dykstra/DFRWS_Dykstra.pdf

Embleton, S., Sparks, S., & Zou, C. (2008). SMM Rootkits: A new breed of OS independent malware. Paper present at Proceedings of the 4ᵗʰ International Conference on Security and Privacy in Communication Networks (SecureComm). Istanbul, Turkey. 1-12. doi:10.1145/1460877.1460892

Ferrie, P. (2006). Attacks on virtual machine emulators. Retrieved on January 14, 2014 from http://www.symantec.com/avcenter/reference/Virtual_Machine_Threats.pdf

F-Response. Remote analysis capability for X-Ways Forensics. Retrieved on January 14, 2014 from http://www.x-ways.net/forensics/f-response.html

GMG Systems. (2013). KnTTools with KnTList. Retrieved on January 14, 2014 from http://gmgsystemsinc.com/knttools

Goel, S. (2009). Digital forensics and cyber crime. Paper presented at First International ICST Conference, Albany, NY, USA, Sept 30 - Oct 2, 2009.

Graham J., Howard R., & Olson. R. (2010). Cyber security essentials. Boca Raton, FL: Auerbach Publications.

Graziano, M., Lanzi, A., & Balzarotti, D. (2013). Hypervisor memory forensics. In J.Stolfo, A. Stavrou, V. Wright (Eds.), *Research in Attacks, Intrusions, and Defenses*. Paper presented at The 16ᵗʰ International Symposium, RAID 2013, Rodney Bay, St. Lucia, 23-25 October, 21-40.

Guidance Software. (2013). EnCase forensic. Retrieved on January 14, 2014 from https://www.encase.com/encase-forensic.htm

Halderman, J.A., Schoen, D.D., Heninger, N., Clarkson, W., Paul, W., Calandrino, J.A., Feldman, A.J., ... Felten E.W. (2008). Lest we remember: Cold boot attacks on encryption keys. Paper presented at 17ᵗʰ USENIX Security Symposium, San Jose, CA, July, 45–60.

Haruyama, T., & Suzuki, H. (2012). One-byte modification for breaking memory forensic analysis. Retrieved on January 14, 2014 from http://media.blackhat.com/bh-eu-12/Haruyama/bh-eu-12-Haruyama-Memory_Forensic-Slides.pdf

Hay, A. F. (2012). Forensic memory analysis for Apple OS X. (Master's thesis). Retrieved from NTIS. (ADA562777)

HBGary. (2013). FastDump. Retrieved on January 14, 2014 from http://hbgary.com/free_tools

Hejazi, S. (2009). Analysis of Windows memory for forensic investigations. (Master's thesis). Retrieved on January 14, 2014 from http://spectrum.library.concordia.ca/976393/1/MR63196.pdf

Hoglund, G. (2011). A brief history of physical memory forensics, Retrieved on January 14, 2014 from http://fasthorizon.blogspot.ru/2011/05/brief-history-of-physical-memory.html

Hoglund, G., & Butler, J. (2005). *Rootkits: Subverting the Windows Kernel*. Massachusetts, US: Addison-Wesley Professional.

Johannes, S., & Michael, C. (2013). Anti-forensic resilient memory acquisition, Retrieved on January 14, 2014 from http://dfrws.org/2013/proceedings/DFRWS2013-13.pdf

Klein, T. (2013). Process dumper. Retrieved on January 14, 2014 from http://www.trapkit.de/research/forensic/pd

Komal, B. (2013, October 1). Step by step memory forensics. *eForensics Magazine*, *15*(19), pp. 20-35.

Korkin, I. (2012). Windows 8 is cyber-battlefield. Retrieved on January 14, 2014 from www.igorkorkin.blogspot.com/2012/09/windows-8-is-cyber-battlefield.html

Korkin, I. (2012) Anti-Rootkits in the era of cyber wars. *Hakin9 Extra Magazine* (English Edition), *2*(7), 26-29. 07/2012 (11) ISSN 1733-7186.

Korkin, I. (2013). Windows NT4.0 source code. Retrieved on January 14, 2014 from http://igorkorkin.blogspot.ru/2013/09/windows-nt-40-full-free-source-code-912_16.html

Kuhn, S., & Taylor, S. (2012). A forensic hypervisor for process tracking and exploit discovery. Paper present at Military Communications Conference, MILCOM, Orlando, FL, Oct 29-Nov 1, 2012,1-5. doi:10.1109/MILCOM.2012.6415817

Lawson, N. (2007). Don't tell Joanna. The virtualized rootkit is dead. Retrieved on January 14, 2014 from http://www.matasano.com/research/bh-usa-07-ptacek_goldsmith_and_lawson.pdf

Lin, Z., Rhee, J., Zhang, X., Xu, D., & Jiang, X. (2011). SigGraph: Brute force scanning of kernel data structure instances using graph-based signatures. Paper presented at the 17th Annual Network and Distributed System Security Symposium (NDSS), CA, 28 February, 1-18.

Linchpin Labs (2010). ATSIV utility. Retrieved on January 14, 2014 from http://www.linchpinlabs.com

Lioy, A., Ramunno, G., & Vernizzi, D. (2009). Trusted-computing technologies for the protection of critical information systems. Paper presented at the International Workshop on Computational Intelligence in Security for Information Systems CISIS'08, Burgos, Spain, 23-26 September, 77-83. Berlin: Springer.

Mandiant. (2009). Software downloads memoryze. Retrieved on January 14, 2014 from https://www.mandiant.com/resources/download/memoryze

ManTech Int. (2009). Sourceforge MDD. Retrieved on January 14, 2014 from http://sourceforge.net/projects/mdd

Milkovic, L. (2012). Defeating Windows memory forensics. Retrieved on January 14, 2014 from http://events.ccc.de/congress/2012/Fahrplan/events/5301.en.html

Moomsols. (2009). DumpIt. Retrieved on January 14, 2014 from http://www.moonsols.com/

MSDN. (2010). Windows research kernel source code. Retrieved on January 14, 2014 from https://www.microsoft.com/education/facultyconnection/articles/articledetails.aspx?cid=2416&c1=en-us&c2=0

MSDN. (2009) XADM: How to use userdump.exe to capture the state of the information store. Retrieved on January 14, 2014 from http://support.microsoft.com/kb/250509/en-us

MSDN. (2013). Forcing a system crash from the keyboard. Retrieved on January 14, 2014 from http://msdn.microsoft.com/en-us/library/windows/hardware/ff545499(v=vs.85).aspx

Okolica, J. & Peterson, G. (2011). Extracting forensic artifacts from Windows O/S memory. Retrieved on January 14, 2014 from http://ie.archive.ubuntu.com/disk1/disk1/download.sourceforge.net/pub/sourceforge/c/cm/cmat/CMAT%20Technical%20Report.pdf

Okolica, J., & Peterson, G. (2010). A compiled memory analysis tool. *IFIP Advances in Information and Communication Technology*, *337*, 195-204. doi:10.1007/978-3-642-15506-2_14

Patel A., & Mistry N. (2013). An analyzing of different techniques and tools to recover data from volatile memory. *International Journal for Scientific Research & Development*, *1*(2), 219-225.

ReactOS. (2013). ReactOS source code. Retrieved on January 14, 2014 from http://doxygen.reactos.org

Reina, A., Fattori, A., Pagani, F., Cavallaro, L., & Bruschi, D. (2012). When hardware meets software: A bulletproof solution to forensic memory acquisition. Proceedings of the 28th Annual Computer Security Applications Conference (ACSAC), NY, USA, 79-88. doi:10.1145/2420950.2420962

Reuben, J. (2007). A survey on virtual machine security. Retrieved on January 14, 2014 from http://www.tml.tkk.fi/Publications/C/25/papers/Reuben_final.pdf

Ruff, N. (2007). Windows memory forensics. *Journal of Computer Virology and Hacking Techniques*, *4*(2), 83-100. doi:10.1007/s11416-007-0070-0

Rutkowska J. (2007). Beyond the CPU: Defeating hardware based RAM acquisition (part I: AMD case). Retrieved from http://www.first.org/conference/2007/papers/rutkowska-joanna-slides.pdf

Rutkowska, J. (2005). Thoughts about cross-view based Rootkit detection. Retrieved on January 14, 2014 from http://es.thehackademy.net/madchat/vxdevl/library/Thoughts%20about%20Cross-View%20based%20Rootkit%20Detection.pdf

Rutkowska, J. (2006). Introducing stealth malware taxonomy. Retrieved on January 14, 2014 from http://theinvisiblethings.blogspot.ru/2006/11/introducing-stealth-malware-taxonomy.html

Rutkowska, J., & Tereshkin, (2007). A. IsGameOver(). Anyone?. Retrieved on January 14, 2014 from http://www.blackhat.com/presentations/bh-usa-07/Rutkowska/Presentation/bh-usa-07-rutkowska.pdf

Saur, K., & Grizzard, J. (2010). Locating x86 paging structures in memory images. *Journal Digital Investigation*: *The International Journal of Digital Forensics & Incident Response*. *7*(1), 28-37. doi:10.1016/j.diin.2010.08.002

Schatz, B. (2007). BodySnatcher: Towards reliable volatile memory acquisition by software. *Journal digital investigation*: *The International Journal of Digital Forensics & Incident Response*, *4*, 126–134. doi:10.1016/j.diin.2007.06.009

Schuster, A. (2006). Searching for processes and threads in Microsoft Windows memory dumps. *Journal Digital Investigation*: *The International Journal of Digital Forensics & Incident Response, 3*. 10-16 doi:10.1016/j.diin.2006.06.010

Shosha, A. F., Chen-Ching, L., Gladyshev, P., & Matten, M. (2012). Evasion-resistant malware signature based on profiling kernel data structure objects. Paper presented at The International Conference on Risks and Security of Internet and Systems (CRiSIS), Cork, 10-12 October, 1-8.

Silakov, D. V. (2012). The use of hardware virtualization in the context of information security, Programming and Computer Software, *38*(5), 276-280. doi:10.1134/S0361768812050064

Sparks, S., & Butler, J. (2005). Shadow walker: Raising the bar for rootkit detection. Retrieved on January 14, 2014 from http://www.blackhat.com/presentations/bh-jp-05/bh-jp-05-sparks-butler.pdf

Stewin, P., & Bystrov I. (2012). Understanding DMA malware. Paper presented at Proceedings of the 9th Conference on Detection of Intrusions and Malware & Vulnerability Assessment, Heraklion, Crete, Greece, 26-27 July, 21-41.

Technology Pathways. (2013). ProDiscover. Retrieved on January 14, 2014 from http://www.techpathways.com/ProDiscoverDFT.htm

Tereshkin, A., & Wojtczuk, R. (2009). Introducing Ring -3 Rootkits. Retrieved on January 14, 2014 from http://www.blackhat.com/presentations/bh-usa-09/TERESHKIN/BHUSA09-Tereshkin-Ring3Rootkit-SLIDES.pdf

Tsaur, W. (2012). Strengthening digital rights management using a new driver-hidden rootkit. *IEEE Transactions on Consumer Electronics*, *58*(2), 479-483. doi: 10.1109/TCE.2012.6227450

Tsaur, W. & Chen, Y. (2010). Exploring Rootkit detectors' vulnerabilities using a new windows hidden driver based Rootkit. Paper presented at The 2nd IEEE International Conference on Social Computing (SocialCom2010), Minneapolis, MN, 20-22 August, 842-848. doi:10.1109/SocialCom.2010.127

Tsaur, W., & Yeh, L. (2012). Identifying Rootkit infections using a new windows hidden-driver-based Rootkit. Paper presented at The International Conference on Security and Management, Las Vegas, USA, 16-19 July, 1-7.

Vasileios, V. (2012). Diving into windows memory forensics. (Master's thesis). Retrieved on January 14, 2014 from http://digilib.lib.unipi.gr/dspace/bitstream/unipi/5564/1/Chatzis-Vovas.pdf

Vasudevan, A. (2008). MalTRAK: Tracking and eliminating unknown malware. Paper presented at Annual Computer Security Applications Conference, Anaheim, CA, 8-12 December, 311-321.

Ververis, V. (2010). Security evaluation of Intel's active management technology. Master thesis. Retrieved on January 14, 2014 from http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/100402-Vassilios_Ververis-with-cover.pdf

Vidstrom, A. (2013). PMDump. Retrieved on January 14, 2014 from http://ntsecurity.nu/toolbox/pmdump/VmWare-suspendapproachVMX-codememdump

Vomel, S., & Freiling, F. (2011). A survey of main memory acquisition and analysis techniques for the windows operating system, *The International Journal of Digital Forensics & Incident Response*, *8*(1), 3-22. doi:10.1016/j.diin.2011.06.002

Vomel, S., & Lenz, H. (2013). Visualizing indicators of Rootkit infections in memory forensics. Paper presented at 7th International Conference on IT Security Incident Management and IT Forensics (IMF), Nuremberg, German, 12-14 March, 122-139.

Wandong, P., Jiang, Y., Jun; C., & Yinshan, L. (2010). A method for hidden process detection based on routines of thread scheduling list. Paper presented at The International Conference on Internet Technology and Applications (iTAP), Wuhan, China, 20-22 August, 1-5.

Wang, J., Zhang, F., Sun, K., & Stavrou, A. (2009). Firmware-assisted memory acquisition and analysis tools for digital forensics. Proceedings International Workshop on Systematic Approaches to Digital Forensic Engineering, Berkeley, California, USA, 26 May, 1-5.

WindowsSCOPE. (2009). Video: Using WindowsSCOPE to reverse engineer and analyze the shadow walker.

Wojtczuk, R., & Rutkowska, J. (2009). Attacking Intel trusted execution technology. Black Hat DC 2009. Retrieved on January 14, 2014 from
http://invisiblethingslab.com/resources/bh09dc/Attacking%20Intel%20TXT%20-%20paper.pdf

Wojtczuk, R., Rutkowska, J., & Tereshkin A. (2009). Another way to circumvent Intel trusted execution technology, Retrieved on January 14, 2014 from
http://invisiblethingslab.com/resources/misc09/Another%20TXT%20Attack.pdf

Wright, C. (2013). Windows memory forensics & memory acquisition. *eForensics Magazine*, 112-118.

Yu, M., Qi, Z., Lin, Q., Zhong, X., Li, B., & Guan H., (2012). Vis: Virtualization enhanced live acquisition for native system, *Journal of Digital Investigation*, *9*(1), 22–33. doi:10.1016/j.diin.2012.04.002

Zhang, R., Wang, L., & Zhang, S. (2009). Windows memory analysis based on KPCR. Paper presented at 5th International Conference on. Information Assurance and Security, Xi'an, China, 18-20 August, 677-680. doi:10.1109/IAS.2009.103

Zhao, Q., Cao, T. (2009). Collecting sensitive information from Windows physical memory. *Journal of Computers*, *4*(1), 3-10.

Zmudzinski, K. (2009). Methods for selecting cores to execute system management interrupts. Retrieved on January 14, 2014 from
http://www.patentimages.storage.googleapis.com/pdfs/US20090172229.pdf