



**UNIVERSITY OF
PIRAEUS**

THESIS

**DIVING INTO WINDOWS MEMORY
FORENSICS**

By

Chatzis – Vovas Vasileios
A.M. : E/08179

Thesis Advisor: Xenakis Christos
Second Reader: Lamprinoudakis Konstantinos
June 2012

MEMORANDUM FOR THE PRESIDENT

THIS PAGE INTENTIONALLY LEFT BLANK

Title

DIVING INTO WINDOWS MEMORY FORENSICS

Author: Chatzis – Vovas Vasileios

A.M. : E/08179

Thesis Advisor: Xenakis Christos

Abstract

During a forensic investigation of a computer system, the ability to retrieve volatile information can be of critical importance. The contents of RAM could reveal malicious code running on the system that has been deleted from the hard drive or, better yet, that was never resident on the hard drive at all. RAM can also provide the programs most recently run and files most recently opened in the system.

However, due to the nature of modern operating systems, these programs and files are not typically stored contiguously—which makes most retrieval efforts of files larger than one page size futile. To date, analysis of RAM images has been largely restricted to searching for ASCII string content, which typically only yields text information such as document fragments, passwords or scripts.

This thesis explores the memory management structures in a Windows system (Mainly Windows Xp and Windows 7) to make sense out of the chaos in RAM and facilitate the retrieval of files/programs larger than one page size. The analysis includes methods for incorporating swap space information for files that may not reside completely within physical memory.

The results of this thesis will become the basis of later research efforts in RAM forensics. This includes the creation of tools that will provide forensic analysts with a clear map of what is resident in the volatile memory of a system.

ACKNOWLEDGMENTS

I would like to thank everyone who contributed in the production of this undergraduate thesis, which proved to be one of the most important parts of my whole studies by giving me the chance to enrich my knowledge and experience on the general area of Forensics.

I would also like to thank Dr. Christos Xenakis for his input and constructive criticism because it helped improve my comprehension of the subject and technical writing and, therefore, any reader's comprehension of this thesis.

Last but not least I would like to thank my family for the psychological and material support throughout my studies.

MEMORANDUM FOR THE PRESIDENT

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

ABSTRACT.....	3
ACKNOWLEDGMENTS.....	4
TABLE OF CONTENTS.....	6
I. INTRODUCTION.....	9
A. RAM FORENSICS BACKGROUND.....	10
B. PURPOSE OF STUDY.....	10
C. THESIS ORGANIZATION.....	11
II. BACKGROUND.....	13
A. PARAMETERS OF INVESTIGATION.....	13
B. TOOLS USED IN THE INVESTIGATION.....	13
III. CURRENT STATE OF RAM FORENSICS.....	16
IV. ANALYSIS.....	18
A. OVERVIEW.....	18
B. ANALYZING A ZEUS BOT INFECTED SYSTEM.....	19
C. ANALYZING A STUXNET INFECTED SYSTEM.....	28
D. PULLING PASSWORDS FROM A MEMORY DUMP.....	31
E. CARVING FILES FROM A MEMORY DUMP.....	33
V. CONCLUSION.....	34
A. SUMMARY.....	34
B. PROBLEMS.....	34
C. FUTURE WORK.....	34
LIST OF REFERENCES.....	36

MEMORANDUM FOR THE PRESIDENT

THIS PAGE INTENTIONALLY LEFT BLANK

ЗАВЕЩАНИЕ ПЕРВА

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Computer-aided crime has been a significant problem for industry in recent years. The FBI estimated the financial losses related to computer incidents in the United States at 72.2 billion for the year 2011.

These incidents included (but were not limited to): viruses, worms, financial fraud, network intrusion, and the sabotage of data or networks. Computer crime has also posed a threat to national security.

Credit card information stolen from compromised systems might have been used to fund terrorist activities. As criminals become more adept at breaking laws through the use of computers, law enforcement agents must hone their ability to investigate these types of cases. Computer Forensics can be used to establish who committed the crime and to reconstruct how the crime was executed.

One specific branch of forensics gaining momentum concerns itself with RAM analysis. Traditionally, when a forensic investigation is performed on a computer of interest, one of the first things done is to gather any volatile information that can be gleaned from the victim system. Sometimes this includes making a copy of the system's RAM content, which is analyzed with simple searches for ASCII or Unicode string content because few tools exist and few people are trained to perform a more in-depth analysis of the memory dump [Ref. 1].

Some of the key pieces of volatile information that a forensic analyst is looking for are the currently running processes of a system and the files most recently used. An educated investigation of a RAM dump could yield this information. Some might ask why it is a good idea to use such a technique if there are programs available (such as ps1) that will enumerate running processes. The answer is that these programs can be subverted if the system they are running on is compromised with a loadable kernel module rootkit—a piece of malware that can manipulate the execution of system commands.

In addition, advanced malware techniques allow for the injection of malicious code directly into running processes such that no new process is visible to standard tools. A forensic exploration of physical memory can look at kernel structures directly and, consequently see through any such deceptions.

Recent worms do not write any data to disk. All data remains in physical memory. This renders standard disk forensics useless and becomes yet another reason why thoroughly inspecting RAM is a growing necessity. It may be the only way to directly detect the presence of malware and give an investigator an opportunity to retrieve full and accurate information from a compromised system.

A. RAM FORENSICS BACKGROUND

The field of computer forensics is young. The FBI created a Computer Analysis and Response team (CART) in 1984—which did not become fully functional until 1991—to supplement its well-established investigation protocols for terrorism and violent. Since then, other public and private organizations have followed suit and now, years later, forensics is beginning to take shape.

Within the forensics community, a large share of attention has been paid to analyzing non-volatile media such as hard drives or storage peripherals. More recently the rise of networks has created an interest in the study of network-based evidence as well. Both of these subjects have existing, extensive bodies of knowledge. This is not the case for RAM analysis. The analysis of volatile memory is such a young area, in fact, that one is hard pressed to find more than one paper directly addressing analysis of Linux RAM contents. As an example of the lack of attention to this critical need, the popular book *Incident Response & Computer Forensics* devotes 7 lines of coverage to RAM analysis in a twenty-two-page chapter devoted to live data collection from Unix systems.

RAM analysis, like all other forensic endeavors, is concerned with the retrieval of information that can serve as evidence in criminal investigations. More specifically, it is the attempt to use memory management structures in computers as maps to extract files and executables resident in a computer's physical memory. These files/executables can be used to prove that a crime has transpired or to trace how it came to pass. The usefulness of this type of investigation lies in the fact that any information found in RAM is known to have been recently running on the victim system. Additionally, volatile memory examination can stand up to conventional attempts at thwarting forensic efforts—such as function hooking which is a way to attach a chosen function to the normal flow of control in a computer system. For example, if a rootkit has hooked itself into the Linux kernel and is intercepting calls to ps, it can exclude whatever process it wants to hide from the returned list of processes.

B. PURPOSE OF STUDY

The immediate purpose of this research is to discover what forensic techniques can be used effectively on the physical memory of a Windows system running the XP or 7 edition. Some techniques for volatile memory forensics have been developed for the XP but they have not yet been tested successfully in the Windows7 version in which some of the fundamental structures involved in memory management have been modified.

The more general goal of this research is to improve the methods of analyzing RAM dumps. Currently, the typical way to analyze physical memory on a computer is to run a string search on the entire memory image in the hopes of finding information such as passwords, the cleartext of a recently typed encrypted message, or the contents of a file.

Unfortunately, during this type of search, valuable context information is lost. For example, it becomes impossible to determine whether recovered string fragments represent the contents of executable files, data files, or runtime program data. This is an unsophisticated “stab in the dark” type of analysis that can only yield a small amount of useful information—

an unfortunate result when the contents of physical memory are a rich source of forensic evidence. As criminals become more adept at creating malware that can elude current methods of digital forensic investigation, forensics methods must evolve to meet the challenge.

When the author of a piece of malware decides to design it to reside exclusively in physical memory—and thereby evade any hard drive investigation—the forensic analyst must have a way to detect it. The goal of this research is to provide the basis for the development of tools that the forensic analyst can use in a detailed analysis of Windows memory images.

C. THESIS ORGANIZATION

This paper will present forensic techniques and the usage of tools that will be able to extract files and executables stored in a computer's physical memory.

Chapter II will detail the programs used in the process of acquisition on the memory images.

Chapter III will discuss the current state of RAM forensics.

Chapter IV will provide a description of the analysis performed on a Windows system.

The analysis consists of two parts. The first is placing specific files in memory, imaging the memory, and seeing if the file in question can be retrieved. The second is to see what other useful information can be extracted from the memory image using tools on a linux system to analyze our pre-acquired memory image.

Chapter V will summarize the results of this thesis and problems encountered along the way. Additionally, the chapter will describe what future work can be performed in the field of RAM forensics.

ЗАВЕЩАНИЕ ПЕРВА

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND

This section describes all of the major components of the Linux virtual memory management system. A short description of each component is provided—as are the locations of the corresponding definitions in the Linux source code. The base directory location assumed throughout is `/usr/src/Linux-2.6.13-15`. This was the default location created by the SUSE 10 distribution used in this thesis

Please note that depending on the version of the Linux kernel used, the version number will vary.

A. PARAMETERS OF INVESTIGATION

The research conducted in this paper was performed on Windows XP and 7 system pre-acquired memory images.

The amount of RAM was deliberately chosen to ease the burden of translating from virtual to physical memory addresses by avoiding high memory (> 896 MB) translations. Use of high memory would not allow for the simple memory conversion scheme outlined in section C of this chapter.

Additional tools used during the course of this research are as outlined below.

B. TOOLS USED IN THE INVESTIGATION

1. DumpIt

MoonSols DumpIt is a fusion of win32dd and win64dd in one executable, no options is asked to the end-user. Only a double click on the executable is enough to generate a copy of the physical memory in the current directory.

2. Foremost

Foremost is a console program to recover files based on their headers, footers, and internal data structures. This process is commonly referred to as data carving. Foremost can work on image files, such as those generated by dd, Safeback, Encase, etc, or directly on a drive. The headers and footers can be specified by a configuration file or you can use command line switches to specify built-in file types. These built-in types look at the data structures of a given file format allowing for a more reliable and faster recovery.

3. Volatility

The Volatility Framework is a completely open collection of tools, implemented in Python under the GNU General Public License, for the extraction of digital artifacts from volatile memory (RAM) samples. The extraction techniques are performed completely independent of the system being investigated but offer unprecedented visibility into the runtime state of the system.

4. Strings

For each *file* given, GNU **strings** prints the printable character sequences that are at least 4 characters long (or the number given with the options below) and are followed by an unprintable character. By default, it only prints the strings from the initialized and loaded sections of object files; for other types of files, it prints the strings from the whole file.

5. Graphviz

Graphviz is an open source graph visualization software. It basically takes an (textual) input file (for example this dot file) that declaratively describes the graph and converts it into a viewable output format (such as bmp, gif, ps etc).

6. DD

DD is a common unix program whose primary purpose is the low-level copying and conversion of raw data. Here it is used to copy or convert memory images (for example from bin to raw or from vmem to dmp).

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΡΠΑ

THIS PAGE INTENTIONALLY LEFT BLANK

III. CURRENT STATE OF MEMORY FORENSICS

Ram forensics is a blossoming field and, as such, has not yet reached maturity. In fact, it can be safely said that this field is in its infancy stage.

More and more scientific documents start to describe the collection procedure for volatile memory in the most general of terms. They all agree that an investigator handling an incident should begin by collecting volatile data, which includes physical memory.

Unfortunately, they don't specify how an investigator should approach the analysis of a system's memory. Just a handful of books mentions some useful programs such as ps that can be used to extract volatile data from a system. One incident response text only goes as far as mentioning that few people go further than running a string search of a memory image.

On the Windows side, the latest demonstrations of physical memory forensics techniques can be found in the solutions to the 2005 Memory Analysis Challenge presented by the Digital Forensic Research Workshop website. .

They distributed two memory images and asked researchers to answer a number of questions about a security incident. The challenge produced two seminal works. The first, by Chris Betz, introduced a tool called memparser. Second, by George Garner and Robert-Jan Mora produced KnTList.

Some of the theory discussed by the two winning answers can be applied to a Linux investigation and some cannot.

At the Blackhat Federal conference in March 2007, Aaron Wlaters and Nick Petroni released a suite called volatools.

Although it only worked on Windows XP Service Pack 2 images, it was able to produce a number of useful data. Volatools was updated and re-released as Volatility in August 2007, and is now maintained and distributed by Volatile Systems.

ЗАВЕЩАНИЕ ПЕРВА

THIS PAGE INTENTIONALLY LEFT BLANK

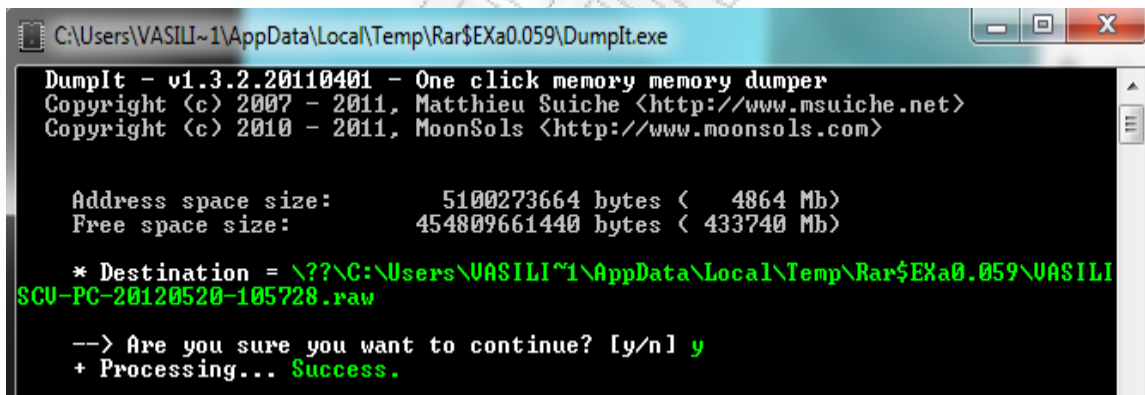
IV. ANALYSIS

A. OVERVIEW

The next few sections go through specific examples of how to perform certain analysis methods on the physical memory of a Windows system. However, this section offers the reader a blueprint of those methods so that s/he can better understand them when they are explained in detail.

The first thing we need to do when we want to perform a memory acquisition of a Windows or Linux System is to dump the memory of the system to a memory image so that we can examine it using a number of tools.

DumpIt by MoonSols is a pretty automated memory dumper that works on all versions of Windows. It's pretty straight forward since it's a one- click version of the general MoonSols project.



```
C:\Users\VASIL~1\AppData\Local\Temp\Rar$EXa0.059\DumpIt.exe
DumpIt - v1.3.2.20110401 - One click memory memory dumper
Copyright (c) 2007 - 2011, Matthieu Suiche <http://www.msuiche.net>
Copyright (c) 2010 - 2011, MoonSols <http://www.moonsols.com>

Address space size:      5100273664 bytes ( 4864 Mb)
Free space size:        454809661440 bytes ( 433740 Mb)

* Destination = \\?\C:\Users\VASIL~1\AppData\Local\Temp\Rar$EXa0.059\VASIL~1\SCU-PC-20120520-105728.raw
--> Are you sure you want to continue? [y/n] y
+ Processing... Success.
```

Now we have our system memory image in .raw extension, something we can change if we convert it to another desirable extension with the DD program in a linux environment.

So we move our acquired image to a linux box for further analyzing and testing, using the tools we listed on chapter 3.

B. ANALYZING A ZEUS BOT INFECTED SYSTEM

The developers of Volatility project have provided a sample image that's infected with Zeus for us to practice on.

So we navigate to volatility folder and run `volatiliy.py` (python written program) And we specify the image file we want to examine with the `(-f)` argument and we use the `(ImageInfo)` plugin, so that we can extract information about the image.

```
root@bt:/pentest/forensics/volatility# ./vol.py -f ~/Desktop/zeus.vmem imageinfo
Volatile Systems Volatility Framework 2.1_alpha
Determining profile based on KDBG search...
```

Suggested Profile(s) : WinXPSP2x86, WinXPSP3x86 (Instantiated with WinXPSP2x86)

```
AS Layer1 : JKIA32PagedMemoryPae (Kernel AS)
AS Layer2 : FileAddressSpace (/root/Desktop/zeus.vmem)
PAE type : PAE
DTB : 0x319000
KDBG : 0x80544ce0L
KPCR : 0xffdff000L
KUSER_SHARED_DATA : 0xffdf0000L
Image date and time : 2010-08-15 19:17:56 UTC+0000
Image local date and time : 2010-08-15 15:17:56 -0400
Number of Processors : 1
Image Type : Service Pack 2
```

Alright, so we can see that this is a XP SP2 image on a 32-bit system, so lets move further along by using the `(pslist)` plugin to determine the processes that were running on the system.

```
root@bt:/pentest/forensics/volatility# ./vol.py -f ~/Desktop/zeus.vmem --profile
WinXPSP2x86 pslist
```

Volatile Systems Volatility Framework 2.1_alpha

Offset(V)	Name	PID	PPID	Thds	Hnds	Time
0x810b1660	System	4	0	58	379	1970-01-01 00:00:00
0xff2ab020	smss.exe	544	4	3	21	2010-08-11 06:06:21
0xff1ecda0	csrss.exe	608	544	10	410	2010-08-11 06:06:23
0xff1ec978	winlogon.exe	632	544	24	536	2010-08-11 06:06:23
0xff247020	services.exe	676	632	16	288	2010-08-11 06:06:24
0xff255020	lsass.exe	688	632	21	405	2010-08-11 06:06:24
0xff218230	vmacthlp.exe	844	676	1	37	2010-08-11 06:06:24
0x80ff88d8	svchost.exe	856	676	29	336	2010-08-11 06:06:24
0xff217560	svchost.exe	936	676	11	288	2010-08-11 06:06:24
0x80fbf910	svchost.exe	1028	676	88	1424	2010-08-11 06:06:24

0xff22d558	svchost.exe	1088	676	7	93	2010-08-11 06:06:25
0xff203b80	svchost.exe	1148	676	15	217	2010-08-11 06:06:26
0xff1d7da0	spoolsv.exe	1432	676	14	145	2010-08-11 06:06:26
0xff1b8b28	vmtoolsd.exe	1668	676	5	225	2010-08-11 06:06:35
0xff1fdc88	VMUpgradeHelper	1788	676	5	112	2010-08-11 06:06:38
0xff143b28	TPAutoConnSvc.e	1968	676	5	106	2010-08-11 06:06:39
0xff25a7e0	alg.exe	216	676	8	120	2010-08-11 06:06:39
0xff364310	wscntfy.exe	888	1028	1	40	2010-08-11 06:06:49
0xff38b5f8	TPAutoConnect.e	1084	1968	1	68	2010-08-11 06:06:52
0x80f60da0	wuauclt.exe	1732	1028	7	189	2010-08-11 06:07:44
0xff3865d0	explorer.exe	1724	1708	13	326	2010-08-11 06:09:29
0xff3667e8	VMwareTray.exe	432	1724	1	60	2010-08-11 06:09:31
0xff374980	VMwareUser.exe	452	1724	8	207	2010-08-11 06:09:32
0x80f94588	wuauclt.exe	468	1028	4	142	2010-08-11 06:09:37
0xff224020	cmd.exe	124	1668	0	-----	2010-08-15 19:17:55

Nothing immediately stands out as they all look legitimate processes that are running on the box. Let's see if any of the are hiding with a new command out of 2.1 Alpha volatility version which is psxview.

```
root@bt:/pentest/forensics/volatility# ./vol.py -f ~/Desktop/zeus.vmem --profile WinXPSP2x86 psxview
```

Volatile Systems Volatility Framework 2.1_alpha

Offset	Name	Pid	pslist	psscan	thrdproc	pspcid	csrss
0x06499b80	svchost.exe	1148	1	1	1	1	1
0x04b5a980	VMwareUser.exe	452	1	1	1	1	1
0x05f027e0	alg.exe	216	1	1	1	1	1
0x0655fc88	VMUpgradeHelper	1788	1	1	1	1	1
0x0211ab28	TPAutoConnSvc.e	1968	1	1	1	1	1
0x04c2b310	wscntfy.exe	888	1	1	1	1	1
0x061ef558	svchost.exe	1088	1	1	1	1	1
0x06945da0	spoolsv.exe	1432	1	1	1	1	1
0x05471020	smss.exe	544	1	1	1	1	0
0x069d5b28	vmtoolsd.exe	1668	1	1	1	1	1
0x06384230	vmacthlp.exe	844	1	1	1	1	1
0x010f7588	wuauclt.exe	468	1	1	1	1	1
0x066f0da0	csrss.exe	608	1	1	1	1	0
0x010c3da0	wuauclt.exe	1732	1	1	1	1	1
0x06238020	cmd.exe	124	1	1	0	1	0
0x06015020	services.exe	676	1	1	1	1	1
0x04a065d0	explorer.exe	1724	1	1	1	1	1
0x049c15f8	TPAutoConnect.e	1084	1	1	1	1	1
0x0115b8d8	svchost.exe	856	1	1	1	1	1
0x01214660	System	4	1	1	1	1	0
0x01122910	svchost.exe	1028	1	1	1	1	1

0x04be97e8 VMwareTray.exe	432	1	1	1	1	1
0x05f47020 lsass.exe	688	1	1	1	1	1
0x063c5560 svchost.exe	936	1	1	1	1	1
0x066f0978 winlogon.exe	632	1	1	1	1	1
0x069a7328 VMip.exe	1944	0	1	0	0	0

This command (psxview) uses multiple methods for looking at processes artifacts in memory. If any process has (0's) for psscan, plist and thrdproc it's an attempt to hide the process by DKOM (Direct Kernel Object Manipulation). Unfortunately nothing stand out here either so we move on testing some internet connections with the (connections) plugin.

```
root@bt:/pentest/forensics/volatility# ./vol.py -f ~/Desktop/zeus.vmem --profile
```

```
WinXPSP2x86 connections
```

```
Volatile Systems Volatility Framework 2.1_alpha
```

```
Offset(V) Local Address Remote Address Pid
```

```
-----
```

No active connections at the time the dump was taken. So lets scan for connections that may have been previously closed with the (connscan) plugin.

```
root@bt:/pentest/forensics/volatility# ./vol.py -f ~/Desktop/zeus.vmem --profile
```

```
WinXPSP2x86 connscan
```

```
Volatile Systems Volatility Framework 2.1_alpha
```

```
Offset(P) Local Address Remote Address Pid
```

```
-----
```

```
0x02214988 172.16.176.143:1054 193.104.41.75:80 856
```

```
0x06015ab0 0.0.0.0:1056 193.104.41.75:80 856
```

There it is! We have 2 connections here that look to be listed to PID 856, which is SVChost something that is odd. Let's see where these connections are located. A whois report reveals that the IP is located in Moldova.

IP Address	193.104.41.75
Host	193.104.41.75
Location	🇲🇩 MD, Moldova, Republic of
City	-, - -
Organization	PE Voronov Evgen Sergiyovich
ISP	PE Voronov Evgen Sergiyovich

It's well known that a lot of malware calls Eastern Europe and Asia home. So this is pretty suspicious but since it looks like all our processes appear legitimate we might be facing some malware that utilizes code injection. To detect these type of processes MHL has released a plugin called (Malfind). It will detect injected processes so lets run that on our target image.

```
root@bt:/pentest/forensics/volatility# ./vol.py -f ~/Desktop/zeus.vmem --profile WinXPSP2x86 malfind -D ~/Desktop/zeusmalfind
```

```
Process: System Pid: 4 Address: 0x1a0000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 38, MemCommit: 1, PrivateMemory: 1, Protection: 6
```

```
0x001a0000 4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00 MZ.....
0x001a0000 b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 .....@.....
0x001a0000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x001a0000 00 00 00 00 00 00 00 00 00 00 00 00 00 d0 00 00 00 .....
```

```
0x1a0000 4d      DEC EBP
0x1a0001 5a      POP EDX
0x1a0002 90      NOP
0x1a0003 0003    ADD [EBX], AL
0x1a0005 0000    ADD [EAX], AL
0x1a0007 000400  ADD [EAX+EAX], AL
0x1a000a 0000    ADD [EAX], AL
0x1a000c ff      DB 0xff
0x1a000d ff00   INC DWORD [EAX]
0x1a000f 00b800000000  ADD [EAX+0x0], BH
0x1a0015 0000    ADD [EAX], AL
0x1a0017 004000  ADD [EAX+0x0], AL
0x1a001a 0000    ADD [EAX], AL
0x1a001c 0000    ADD [EAX], AL
```

```

0x1a001e 0000    ADD [EAX], AL
0x1a0020 0000    ADD [EAX], AL
0x1a0022 0000    ADD [EAX], AL
0x1a0024 0000    ADD [EAX], AL
0x1a0026 0000    ADD [EAX], AL
0x1a0028 0000    ADD [EAX], AL
0x1a002a 0000    ADD [EAX], AL
0x1a002c 0000    ADD [EAX], AL
0x1a002e 0000    ADD [EAX], AL
0x1a0030 0000    ADD [EAX], AL
0x1a0032 0000    ADD [EAX], AL
0x1a0034 0000    ADD [EAX], AL
0x1a0036 0000    ADD [EAX], AL
0x1a0038 0000    ADD [EAX], AL
0x1a003a 0000    ADD [EAX], AL
0x1a003c d000    ROL BYTE [EAX], 0x1
0x1a003e 0000    ADD [EAX], AL

```

[snip] (about 40 pages)

There is a lot of output so it looks like a lot of our processes are injected with malcode. The reason this plugin can find it, is due to the fact of looking for kernel memory structures that work very closely with VirtualAlloc. These memory structures are in a Vad tree and work closely with memory management aspects of the kernel. Also the plugin outputs hexdumps as well as assembly code at the base location of where the injected code was detected.

With all this output from our plugin lets visit the (pstree) plugin so we can get a hierarchical view on how the code injection may have cascaded.

```

root@bt:/pentest/forensics/volatility# ./vol.py -f ~/Desktop/zeus.vmem --profile
WinXPSP2x86 pstree

```

Volatile Systems Volatility Framework 2.1_alpha

WARNING : psycosupport : Deprecation warning: A plugin is making use of profile.add_types

Name	Pid	PPid	Thds	Hnds	Time
0x810B1660:System			4	0	58 379 1970-01-01 00:00:00
. 0xFF2AB020:smss.exe	544	4	3	21	2010-08-11 06:06:21
.. 0xFF1EC978:winlogon.exe	632	544	24	536	2010-08-11 06:06:23
... 0xFF255020:lsass.exe	688	632	21	405	2010-08-11 06:06:24
... 0xFF247020:services.exe	676	632	16	288	2010-08-11 06:06:24
.... 0xFF1B8B28:vmtoolsd.exe	1668	676	5	225	2010-08-11 06:06:35
..... 0xFF224020:cmd.exe	124	1668	0	-----	2010-08-15 19:17:55
.... 0x80FF88D8:svchost.exe	856	676	29	336	2010-08-11 06:06:24
... 0xFF1D7DA0:spoolsv.exe	1432	676	14	145	2010-08-11 06:06:26
... 0x80FBF910:svchost.exe	1028	676	88	1424	2010-08-11 06:06:24
..... 0x80F60DA0:wuauclt.exe	1732	1028	7	189	2010-08-11 06:07:44

```

..... 0x80F94588:wuaucft.exe          468 1028 4 142 2010-08-11 06:09:37
..... 0xFF364310:wscntfy.exe          888 1028 1 40 2010-08-11 06:06:49
.... 0xFF217560:svchost.exe           936 676 11 288 2010-08-11 06:06:24
... 0xFF143B28:TPAutoConnSvc.e        1968 676 5 106 2010-08-11 06:06:39
..... 0xFF38B5F8:TPAutoConnect.e      1084 1968 1 68 2010-08-11 06:06:52
... 0xFF22D558:svchost.exe            1088 676 7 93 2010-08-11 06:06:25
... 0xFF218230:vmacthlp.exe            844 676 1 37 2010-08-11 06:06:24
... 0xFF25A7E0:alg.exe                 216 676 8 120 2010-08-11 06:06:39
... 0xFF203B80:svchost.exe            1148 676 15 217 2010-08-11 06:06:26
.... 0xFF1FDC88:VMUpgradeHelper       1788 676 5 112 2010-08-11
06:06:38
.. 0xFF1ECDA0:csrss.exe                608 544 10 410 2010-08-11 06:06:23
0xFF3865D0:explorer.exe               1724 1708 13 326 2010-08-11 06:09:29
. 0xFF374980:VMwareUser.exe           452 1724 8 207 2010-08-11 06:09:32
. 0xFF3667E8:VMwareTray.exe           432 1724 1 60 2010-08-11 06:09:31

```

We did noticed that services.exe looked to have some code injected into it. Let's take the parent process (winlogon.dmp that was dumped by the malfind) and submit it to VIRUSTOTAL as PID 676 seems to be where the code injection is originated from a hierarchical sence.



SHA256: bfa09ee7a33180135cf2ee2f373b0493f53710e044c27f405e9c681bdb7e4c83

File name: winlogon.exe.66f0978.00ae0000-00b05fff.dmp

Detection ratio: 24 / 42

Analysis date: 2012-05-11 09:52:36 UTC (1 εβδομάδα, 2 ημέρες ago)

[More details](#)

Antivirus	Result	Update
AhnLab-V3	Worm/Win32.IRCBot	20120511
AntiVir	TR/Dropper.Gen	20120511
Antiy-AVL	-	20120511
Avast	Win32.Zbot-BCW [Trj]	20120511
AVG	Win32/Heri	20120510
BitDefender	Gen:Variant.Graftor.22830	20120511
ByteHero	-	20120511
CAT-QuickHeal	Win32.PWS.Zbot.gen!V.4.grp5	20120511
ClamAV	-	20120510
Commtouch	W32/Zbot.AG.gen!Eldorado	20120511
Comodo	TrojWare.Win32.Spy.Zbot.ABW	20120511

24/42 says it's malicious. Seems most of the scans detect it as Zbot. So let's Google around to find some reports and see if we can verify it's presence elsewhere.

“The install function searches for the winlogon.exe” process, allocates some memory within it and decrypts itself into the process”

So it looks like a Zbot/Zeus injects it's code into winlogon.exe. This was apparent after we did our malfind as it detected injected injected code into other processes.

“The bot executable is written to the jand drive as
“C:\WINDOWS\system32\sdra64.exe”.”

So we will use Volatility plugin (filescan) that allows us to identify the file handles that are still hanging around in memory.

```
root@bt:/pentest/forensics/volatility# ./vol.py -f ~/Desktop/zeus.vmem --profile WinXPSP2x86 filescan
```

```
Offset(P) #Ptr #Hnd Access Name
0x00096ca0 1 0 R--r-d \Documents and Settings\Administrator\Start
Menu\Programs\Windows Media Player.lnk
0x00353ad0 1 0 R--rwd \WINDOWS\system32\crypt32.dll
0x00353cb8 1 0 R--rwd \WINDOWS\system32\apphelp.dll
0x003f34f8 3 0 RWD--- \Directory
0x003f3f08 1 0 R--r-d \WINDOWS\system32\ipconf.tsp
[snip]
0x029d9b40 1 1 R----- \WINDOWS\system32\sdra64.exe
0x029d9cf0 1 0 -WD--- \WINDOWS\system32\sdra64.exe
[snip]
```

“The directory “C:\WINDOWS\system32\lowsec\” is created. This directory is not visible in Windows explorer but can be seen from the command line, It's purpose is to contain the following files:

-local.ds: Contains the most recently downloaded DynamicConfig File.

-user.ds: Contains logged information.

-user.ds.lll: temporally created if transmission of logs to the drop server fails. “

These artifacts can also be found in the above file scan to further bolster the case that this is definitely Zeus.

“The Winlogon

(“HKLM\SOFTWARE\Microsoft\WindowsNT\CurrentVersion\Winlogon”)

Registry key's value is appended with the path of the bot executable:

C:\WINDOWS\system32\sdra64.exe. This will cause the bot to execute when the computer restarts.”

Volatility sure enough has a feature to allow us to investigate registry entries. Namely the (printkey) plugin.

```
root@bt:/pentest/forensics/volatility# ./vol.py -f ~/Desktop/zeus.vmem --profile
WinXPSP2x86 printkey -K "Microsoft\Windows NT\CurrentVersion\Winlogon"
Volatile Systems Volatility Framework 2.1_alpha
Legend: (S) = Stable (V) = Volatile
```

```
-----
Registry: \Device\HarddiskVolume1\WINDOWS\system32\config\software
Key name: Winlogon (S)
Last updated: 2010-08-15 19:17:23
```

Subkeys:

- (S) GPEExtensions
- (S) Notify
- (S) SpecialAccounts
- (V) Credentials

Values:

```
REG_DWORD  AutoRestartShell : (S) 1
REG_SZ     DefaultDomainName : (S) BILLY-DB5B96DD3
REG_SZ     DefaultUserName : (S) Administrator
REG_SZ     LegalNoticeCaption : (S)
REG_SZ     LegalNoticeText : (S)
REG_SZ     PowerdownAfterShutdown : (S) 0
REG_SZ     ReportBootOk : (S) 1
REG_SZ     Shell : (S) Explorer.exe
REG_SZ     ShutdownWithoutLogon : (S) 0
REG_SZ     System : (S)
REG_SZ     Userinit : (S)
C:\WINDOWS\system32\userinit.exe,C:\WINDOWS\system32\sdr64.exe,
REG_SZ     VmApplet : (S) rundll32 shell32,Control_RunDLL "sysdm.cpl"
REG_DWORD  SfcQuota : (S) 4294967295
```

[snip]

Well the key is certainly apparent and this is our persistence mechanism. So the Zeus/Zbot injector process is called at start-up to insert it's hooks and malicious code in our legitimate looking process to evade detection. This would be something you'd want to clean up if you were re-mediating the system as well.

“The Windows XP firewall is disabled. This causes a Windows Security Center warning icon to appear in the system tray, the only visible indication that the computer has been infected.”

So lets see if the firewall is up or its has been disabled.

```
root@bt:/pentest/forensics/volatility# ./vol.py -f ~/Desktop/zeus.vmem --profile
WinXPSP2x86 printkey -K
"ControlSet001\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile"
Volatile Systems Volatility Framework 2.1_alpha
Legend: (S) = Stable (V) = Volatile
```

```
-----
Registry: \Device\HarddiskVolume1\WINDOWS\system32\config\system
Key name: StandardProfile (S)
Last updated: 2010-08-15 19:17:24
```

```
Subkeys:
(S) AuthorizedApplications
```

```
Values:
REG_DWORD EnableFirewall : (S) 0
```

So the firewall is currently disabled and if you notice the timestamp on the key as well it looks like this was last updated at 2010-8-15 at 19:17:24.

“ A closer look at its binary file reveals that the spyware was designed to monitor know ZBOT mutexes, _AVIRA_ and _SYSTEM_.”

```
root@bt:/pentest/forensics/volatility# ./vol.py -f ~/Desktop/zeus.vmem --profile
WinXPSP2x86 mutantscan
Volatile Systems Volatility Framework 2.1_alpha
Offset(P) #Ptr #Hnd Signal Thread CID Name
0x000962c0 1 1 1 0x00000000
[snip]
0x06735dc0 2 1 1 0x00000000 _AVIRA_2109
[snip]
```

Well there is certainly a mutex that has been recent in memory for AVIRA which ironically enough is the name of an antivirus engine.

So there we have it, using Volatility we can get a look at a zeus bot infection and determine steps here for possible remediation just based on a memory dump.

C. ANALYZING A STUXNET INFECTED SYSTEM

The developers of Volatility project have provided a sample image that's infected with Zeus for us to practice on.

So we navigate to volatility folder and run `volatiliy.py` (python written program) And we specify the image file we want to examine with the `(-f)` argument and we use the `(ImageInfo)` plugin, so that we can extract information about the image.

```
root@bt:/pentest/forensics/volatility# ./vol.py -f ~/Desktop/stuxnet.vmem imageinfo
Volatile Systems Volatility Framework 2.1_alpha
Determining profile based on KDBG search...
```

Suggested Profile(s) : WinXPSP2x86, WinXPSP3x86 (Instantiated with WinXPSP2x86)

```
AS Layer1 : JKIA32PagedMemoryPae (Kernel AS)
AS Layer2 : FileAddressSpace (/root/Desktop/stuxnet.vmem)
PAE type : PAE
DTB : 0x319000
KDBG : 0x80545ae0L
KPCR : 0xffdf000L
KUSER_SHARED_DATA : 0xffdf000L
Image date and time : 2011-06-03 04:31:36 UTC+0000
Image local date and time : 2011-06-03 00:31:36 -0400
Number of Processors : 1
Image Type : Service Pack 3
```

Alright, so we can see that this is a XP SP3 image on a 32-bit system, so lets move further along by using the `(pslist)` plugin to determine the processes that were running on the system.

```
root@bt:/pentest/forensics/volatility# ./vol.py -f ~/Desktop/stuxnet.vmem --profile
WinXPSP3x86 pslist
```

Volatile Systems Volatility Framework 2.1_alpha

Offset(V)	Name	PID	PPID	Thds	Hnds	Time
0x823c8830	System	4	0	59	403	1970-01-01 00:00:00
0x820df020	smss.exe	376	4	3	19	2010-10-29 17:08:53
0x821a2da0	csrss.exe	600	376	11	395	2010-10-29 17:08:54
0x81da5650	winlogon.exe	624	376	19	570	2010-10-29 17:08:54
0x82073020	services.exe	668	624	21	431	2010-10-29 17:08:54
0x81e70020	lsass.exe	680	624	19	342	2010-10-29 17:08:54
0x823315d8	vmacthlp.exe	844	668	1	25	2010-10-29 17:08:55
0x81db8da0	svchost.exe	856	668	17	193	2010-10-29 17:08:55
0x81e61da0	svchost.exe	940	668	13	312	2010-10-29 17:08:55
0x822843e8	svchost.exe	1032	668	61	1169	2010-10-29 17:08:55

0x81e18b28 svchost.exe	1080	668	5	80	2010-10-29 17:08:55
0x81ff7020 svchost.exe	1200	668	14	197	2010-10-29 17:08:55
0x81fee8b0 spoolsv.exe	1412	668	10	118	2010-10-29 17:08:56
0x81e0eda0 jqs.exe	1580	668	5	148	2010-10-29 17:09:05
0x81fe52d0 vmttoolsd.exe	1664	668	5	284	2010-10-29 17:09:05
0x821a0568 VMUpgradeHelper	1816	668	3	96	2010-10-29 17:09:08
0x8205ada0 alg.exe	188	668	6	107	2010-10-29 17:09:09
0x820ec7e8 explorer.exe	1196	1728	16	582	2010-10-29 17:11:49
0x820ecc10 wscntfy.exe	2040	1032	1	28	2010-10-29 17:11:49
0x81e86978 TSVNCache.exe	324	1196	7	54	2010-10-29 17:11:49
0x81fc5da0 VMwareTray.exe	1912	1196	1	50	2010-10-29 17:11:50
0x81e6b660 VMwareUser.exe	1356	1196	9	251	2010-10-29 17:11:50
0x8210d478 jusched.exe	1712	1196	1	26	2010-10-29 17:11:50
0x82279998 imapi.exe	756	668	4	116	2010-10-29 17:11:54
0x822b9a10 wuauclt.exe	976	1032	3	133	2010-10-29 17:12:03
0x81c543a0 Procmon.exe	660	1196	13	189	2011-06-03 04:25:56
0x81fa5390 wmiprvse.exe	1872	856	5	134	2011-06-03 04:25:58
0x81c498c8 lsass.exe	868	668	2	23	2011-06-03 04:26:55
0x81c47c00 lsass.exe	1928	668	4	65	2011-06-03 04:26:55
0x81c0cda0 cmd.exe	968	1664	0	-----	2011-06-03 04:31:35
0x81f14938 ipconfig.exe	304	968	0	-----	2011-06-03 04:31:35

Looking at this list you can see one of the signs of a Stuxnet infection. There are three copies of lsass.exe running, there should only be one. The lsass process authenticates users for the Winlogon service.

Let's do a process tree list and see if all three instances of lsass correspond to Winlogon:

```

root@bt:/pentest/forensics/volatility# ./vol.py -f ~/Desktop/stuxnet.vmem --profile
WinXPSP3x86 pstree
Volatile Systems Volatility Framework 2.1_alpha
WARNING : psyco.support : Deprecation warning: A plugin is making use of
profile.add_types
Name                Pid  PPid  Thds  Hnds  Time
0x823C8830:System    4    0    59   403   1970-01-01 00:00:00
. 0x820DF020:smss.exe 376   4    3    19   2010-10-29 17:08:53
.. 0x821A2DA0:csrss.exe 600  376   11   395   2010-10-29 17:08:54
.. 0x81DA5650:winlogon.exe 624  376   19   570   2010-10-29 17:08:54
... 0x82073020:services.exe 668  624   21   431   2010-10-29 17:08:54
.... 0x81FE52D0:vmttoolsd.exe 1664  668    5   284   2010-10-29 17:09:05
..... 0x81C0CDA0:cmd.exe 968  1664    0  ----- 2011-06-03 04:31:35
..... 0x81F14938:ipconfig.exe 304  968    0  ----- 2011-06-03 04:31:35
.... 0x822843E8:svchost.exe 1032  668   61  1169   2010-10-29 17:08:55
..... 0x822B9A10:wuauclt.exe 976  1032    3   133   2010-10-29 17:12:03
..... 0x820ECC10:wscntfy.exe 2040  1032    1    28   2010-10-29 17:11:49

```

.... 0x81E61DA0:svchost.exe	940	668	13	312	2010-10-29 17:08:55
.... 0x81DB8DA0:svchost.exe	856	668	17	193	2010-10-29 17:08:55
..... 0x81FA5390:wmiprvse.exe	1872	856	5	134	2011-06-03 04:25:58
... 0x821A0568:VMUpgradeHelper	1816	668	3	96	2010-10-29 17:09:08
... 0x81FEE8B0:spoolsv.exe	1412	668	10	118	2010-10-29 17:08:56
... 0x81FF7020:svchost.exe	1200	668	14	197	2010-10-29 17:08:55
... 0x81C47C00:lsass.exe	1928	668	4	65	2011-06-03 04:26:55
... 0x81E18B28:svchost.exe	1080	668	5	80	2010-10-29 17:08:55
... 0x8205ADA0:alg.exe	188	668	6	107	2010-10-29 17:09:09
... 0x823315D8:vmacthlp.exe	844	668	1	25	2010-10-29 17:08:55
... 0x81E0EDA0:jqs.exe	1580	668	5	148	2010-10-29 17:09:05
... 0x81C498C8:lsass.exe	868	668	2	23	2011-06-03 04:26:55
... 0x82279998:imapi.exe	756	668	4	116	2010-10-29 17:11:54
.. 0x81E70020:lsass.exe	680	624	19	342	2010-10-29 17:08:54
0x820EC7E8:explorer.exe	1196	1728	16	582	2010-10-29 17:11:49
. 0x81C543A0:Procmon.exe	660	1196	13	189	2011-06-03 04:25:56
. 0x81E86978:TSVNCache.exe	324	1196	7	54	2010-10-29 17:11:49
. 0x81E6B660:VMwareUser.exe	1356	1196	9	251	2010-10-29 17:11:50
. 0x8210D478:jusched.exe	1712	1196	1	26	2010-10-29 17:11:50
. 0x81FC5DA0:VMwareTray.exe	1912	1196	1	50	2010-10-29 17:11:50

Looking at the PPID column, you can see that two of the processes connect to PPID 668 and on connects to 624. Looking at the PID you can see that the third instance does in fact tie to Winlogon (624). But the two other instances connect to Services.exe (628). Something is not right.

Let's run the plugin command "malfind" and see what it detects. According to the Volatility Wiki Command Reference, malfind can find hidden or injected code or DLLs in user mode memory. We will run malfind against the whole memory dump and see if it can find any suspicious code.

```
root@bt:/pentest/forensics/volatility# ./vol.py -f ~/Desktop/stuxnet.vmem --profile
WinXPSP3x86 malfind -D ~/Desktop/stuxnetmalfind/
Volatile Systems Volatility Framework 2.1_alpha
Process: explorer.exe Pid: 1196 Address: 0x2550000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 1, MemCommit: 1, PrivateMemory: 1, Protection: 6
```

```
0x02550000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x02550000 00 00 55 02 00 00 00 00 00 00 00 00 00 00 00 ..U.....
0x02550000 10 00 55 02 00 00 00 00 00 00 00 00 00 00 00 ..U.....
0x02550000 20 00 55 02 00 00 00 00 00 00 00 00 00 00 00 ..U.....
```

```
0x2550000 0000      ADD [EAX], AL
0x2550002 0000      ADD [EAX], AL
```

```

0x2550004 0000    ADD [EAX], AL
0x2550006 0000    ADD [EAX], AL
0x2550008 0000    ADD [EAX], AL
0x255000a 0000    ADD [EAX], AL
0x255000c 0000    ADD [EAX], AL
0x255000e 0000    ADD [EAX], AL
0x2550010 0000    ADD [EAX], AL
0x2550012 55      PUSH EBP
0x2550013 0200    ADD AL, [EAX]
0x2550015 0000    ADD [EAX], AL
0x2550017 0000    ADD [EAX], AL
0x2550019 0000    ADD [EAX], AL
0x255001b 0000    ADD [EAX], AL
0x255001d 0000    ADD [EAX], AL
0x255001f 0010    ADD [EAX], DL
0x2550021 005502  ADD [EBP+0x2], DL
0x2550024 0000    ADD [EAX], AL
0x2550026 0000    ADD [EAX], AL
0x2550028 0000    ADD [EAX], AL
0x255002a 0000    ADD [EAX], AL
0x255002c 0000    ADD [EAX], AL
0x255002e 0000    ADD [EAX], AL
0x2550030 2000    AND [EAX], AL
0x2550032 55      PUSH EBP
0x2550033 0200    ADD AL, [EAX]
0x2550035 0000    ADD [EAX], AL
0x2550037 0000    ADD [EAX], AL
0x2550039 0000    ADD [EAX], AL
0x255003b 0000    ADD [EAX], AL
0x255003d 0000    ADD [EAX], AL
0x255003f 00      DB 0x0

```

It finds something it didn't like in explorer.exe right away, but continued to run for quite a while and kicked out two more. Something suspicious in the two copies of lsass.exe – surprise, surprise!

Going to the output directory you see all three suspicious files stored as .dmp files. You can take these files and upload them to VirusTotal to see if it detects anything suspicious. The first explorer.exe file when run against Virus Total did not return anything malicious. But uploading the two lsass files to virus total returns some interesting results. Two virus scanners detected something they don't like in the files. Comodo detects it as a Packed.Win32.MUPX. Gen and VirusBuster detects a Trojan. And indeed there is something fishy there. The real lsass.exe does not have an executable section in its data region, but both of these files do.

That's a very common sign of stuxnet infection.

D. PULLING PASSWORDS FROM A MEMORY DUMP

So lets see how we can find and pull passwords on a windows system from a memory image. We will use the stuxnet image from the example above and try to fetch the admin-guest passwords.

Now, we need the hive list so we can get the starting location in memory of where the registry information resides:

```
root@bt:/pentest/forensics/volatility# ./vol.py -f ~/Desktop/stuxnet.vmem hivelist
Volatile Systems Volatility Framework 2.1_alpha
Virtual   Physical  Name
0xe1069008 0x14b8d008 \Device\HarddiskVolume1\Documents and
Settings\Administrator\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
0xe1077758 0x152b7758 \Device\HarddiskVolume1\Documents and
Settings\Administrator\NTUSER.DAT
0xe1bdb9e8 0x0e1959e8 \Device\HarddiskVolume1\Documents and
Settings\LocalService\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
0xe1bd5b60 0x0e027b60 \Device\HarddiskVolume1\Documents and
Settings\LocalService\NTUSER.DAT
0xe1bc26d8 0x0de626d8 \Device\HarddiskVolume1\Documents and
Settings\NetworkService\Local Settings\Application
Data\Microsoft\Windows\UsrClass.dat
0xe1bb5758 0x0df10758 \Device\HarddiskVolume1\Documents and
Settings\NetworkService\NTUSER.DAT
0xe1628b60 0x0a768b60
\Device\HarddiskVolume1\WINDOWS\system32\config\software
0xe16386b8 0x0a7a06b8
\Device\HarddiskVolume1\WINDOWS\system32\config\default
0xe1638b60 0x0a7a0b60
\Device\HarddiskVolume1\WINDOWS\system32\config\SAM
0xe1628008 0x0a768008
\Device\HarddiskVolume1\WINDOWS\system32\config\SECURITY
0xe13feb60 0x02e6ab60 [no name]
0xe1035b60 0x02a9eb60
\Device\HarddiskVolume1\WINDOWS\system32\config\system
0xe102e008 0x02a98008 [no name]
0x80670a0c 0x00670a0c [no name]
root@bt:/pentest/forensics/volatility# ./vol.py -f ~/Desktop/stuxnet.vmem hashdump -y
0xe1035b60 -s 0xe1638b60 > ~/Desktop/hashstux.txt
```

We now have a list of where several key items are located in the memory dump. Next, we will extract the password hashes from the memory dump. To do this we need to know the starting memory locations for the system and sam keys.

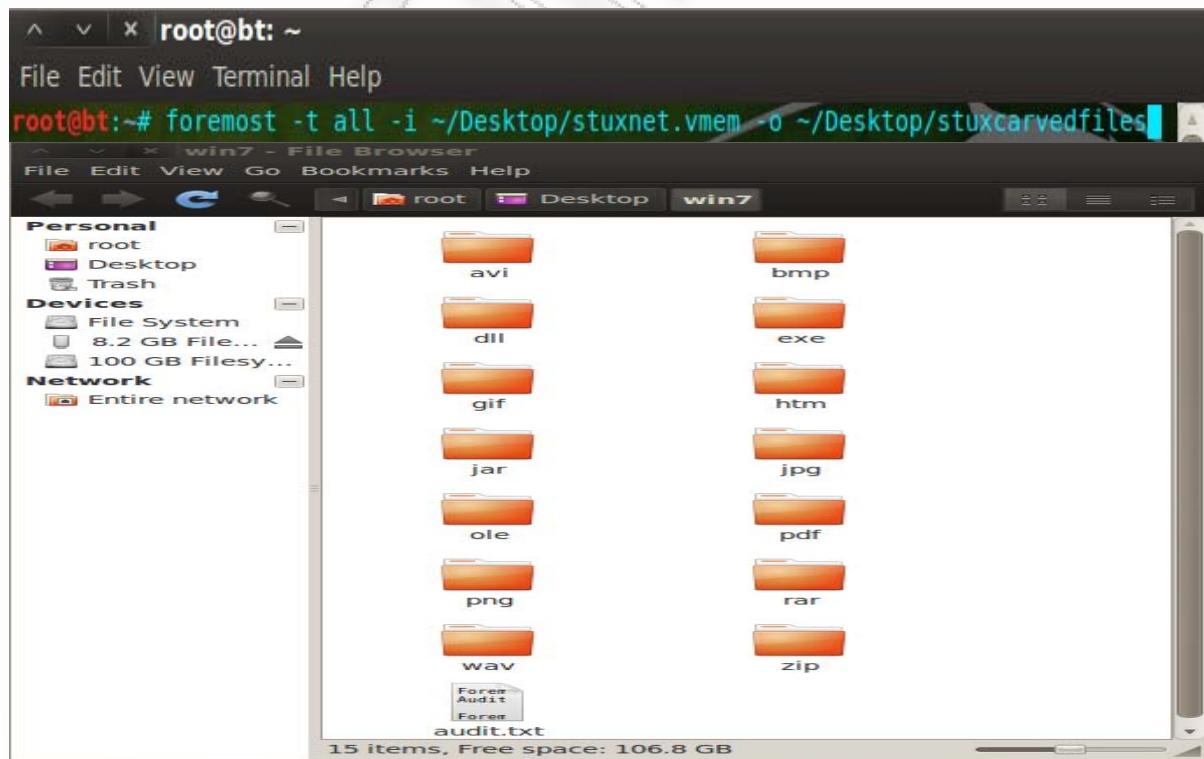
We look in the dump above and copy down the numbers in the first column that correspond to the SAM and SYSTEM locations. Then output the password hashes into a text file called hashes.txt:

```
root@bt:/pentest/forensics/volatility# ./vol.py -f ~/Desktop/stuxnet.vmem -y 0xe1035b60
-s 0xe1638b60 > ~/Desktop/stuxhash.txt
Administrator:500:e52cac67419a9a224a3b108f3fa6cb6d:8846f7eae8fb117ad06bdd830b
7586c:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:
::
HelpAssistant:1000:22d8685792cd2df8392f2d3ec8648d7e:bdedd3a3893c938a7fff9e4e1
234f08a:::
SUPPORT_388945a0:1002:aad3b435b51404eeaad3b435b51404ee:4698b5e815592ec8a1
a8d0073f04320b:::
ASPNET:1003:1b6866646ec27ffe7ac71aa7a7e181b1:3e3397d960245d178c307f19d813a
638:::
```

Now we have the hashes of the Windows Admin, Guest, HelpAssistant, Support and ASPNET password, and by using on them a password cracking tool such as John The Ripper we can find the precise password of the targeted system!

E. CARVING FILES FROM A MEMORY DUMP

We can use the linux program Foremost on our pre-acquired images to “carve” out the files we need or if we want we can make the search general and Foremost will get us All the file that can be extracted from the memory image.



ENVIRONMENTAL PROTECTION

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSION

A. SUMMARY

The memory management changes in the new Windows7 have not adversely affected the forensics techniques developed for the Windows Xp. The discovery of data belonging to a process is possible—as shown in the previous chapter—and should add value to the full exploration of processes that reside in physical memory. The techniques outlined in this paper, or programs based on them should now be integrated into any in-depth forensic analysis of a computer system.

There is a new breed of malware based on the concept that physical memory is largely out of the hands of an examiner. The effects of this insidious software can be mitigated by the application of the findings in this paper. However, unless these analysis techniques are properly automated, many investigators will shy away from performing detailed analysis of physical memory and potentially miss many pieces of evidence vital to such an investigation.

B. PROBLEMS

Throughout the course of the analysis, a few problems were encountered. The first presented itself in the early stages of project development when the assumption was made that the tool were up to date with windows 7 images something that proved to be wrong.

Aside of that we encountered some problems mapping the connections on the memory images we had, something that took quite some time to overcome.

C. FUTURE WORK

The most obvious extension to the work in this paper would be to write a tool that could automate the techniques discussed. This work would include the fine-tuning of the concepts presented here so that they can be translated into the specific language of a program. Such a tool would be a powerful asset to an investigator because tracing hex dumps is not something that many investigators will want to do. That is, the exploration of physical memory would often be skipped in the absence of a good tool to do automate it elegantly and efficiently.

Another future focus of research would be to perform similar analyses on systems with different characteristics. This could take the form of research on other operating systems such as Windows or Solaris. It could also take the form of research on systems running on non-x86 platforms such as PowerPC or 64-bit systems. Systems with more than 896MB of physical memory should be explored as well. Unraveling the process of translating virtual addresses in ZONE_HIGHMEM to physical addresses and back would be very useful as the number of systems with memory in excess of this limit is increasing daily.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

- [1] Prosis, Chris, and Mandia, Kevin, and Pepe, Matt. *Incident Response and Computer Forensics, Second Edition*. McGraw-Hill Osborne Media
- [2] Federal Bureau of Investigation. "History of the FBI, Rise of International Crime: 1980's," <http://www.fbi.gov/libref/historic/history/rise.htm>, 2006
- [3] Burdach, Mariusz. Digital Forensics of the Physical Memory.
- [4] – <http://www.fortiguard.com/analysis/zeusanalysis.html>
- [5] – <http://www.dfrws.org/2007/proceedings/p62-dolan-gavitt.pdf>
- [6] – <http://www.eptuners.com/forensics/contents/examination.htm>
- [7] – http://www.sans.org/reading_room/whitepapers/malicious/clash-titans-zeus-spyeye_33393
- [8] – <http://www.symantec.com/connect/blogs/brief-look-zeusbot-20>
- [9] Gorman, Mel. *Understanding the Linux Virtual Memory Manager (Bruce Perens Open Source)*. Prentice Hall PTR
- [10] Grance, Tim, and Kent, Karen, and Kim, Brian. *NIST Special Publication 800-61: Computer Security Incident Handling Guide*.
- [11] Brezinski, D., and Killalea, T. *RFC 3227: Guidelines for Evidence Collection and Archiving*.
- [12] Burdach, Mariusz. "Finding Digital Evidence in Physical Memory." *2006 Black Hat Federal Conference*. Sheraton Crystal City, Washington DC. 25 January 2006.
- [13] Garner, George M. Jr., and Mora, Robert-Jan "Response to Specific Questions Posed by the DFRWS 2005 Memory Challenge," <http://www.dfrws.org/2005/challenge/index.html>. 6 August 2005.
- [14] Betz, Chris. "DFRWS 2005 Challenge Report," <http://www.dfrws.org/2005/challenge/ChrisBetz-DFRWSChallengeOverview.html>. August 2005.