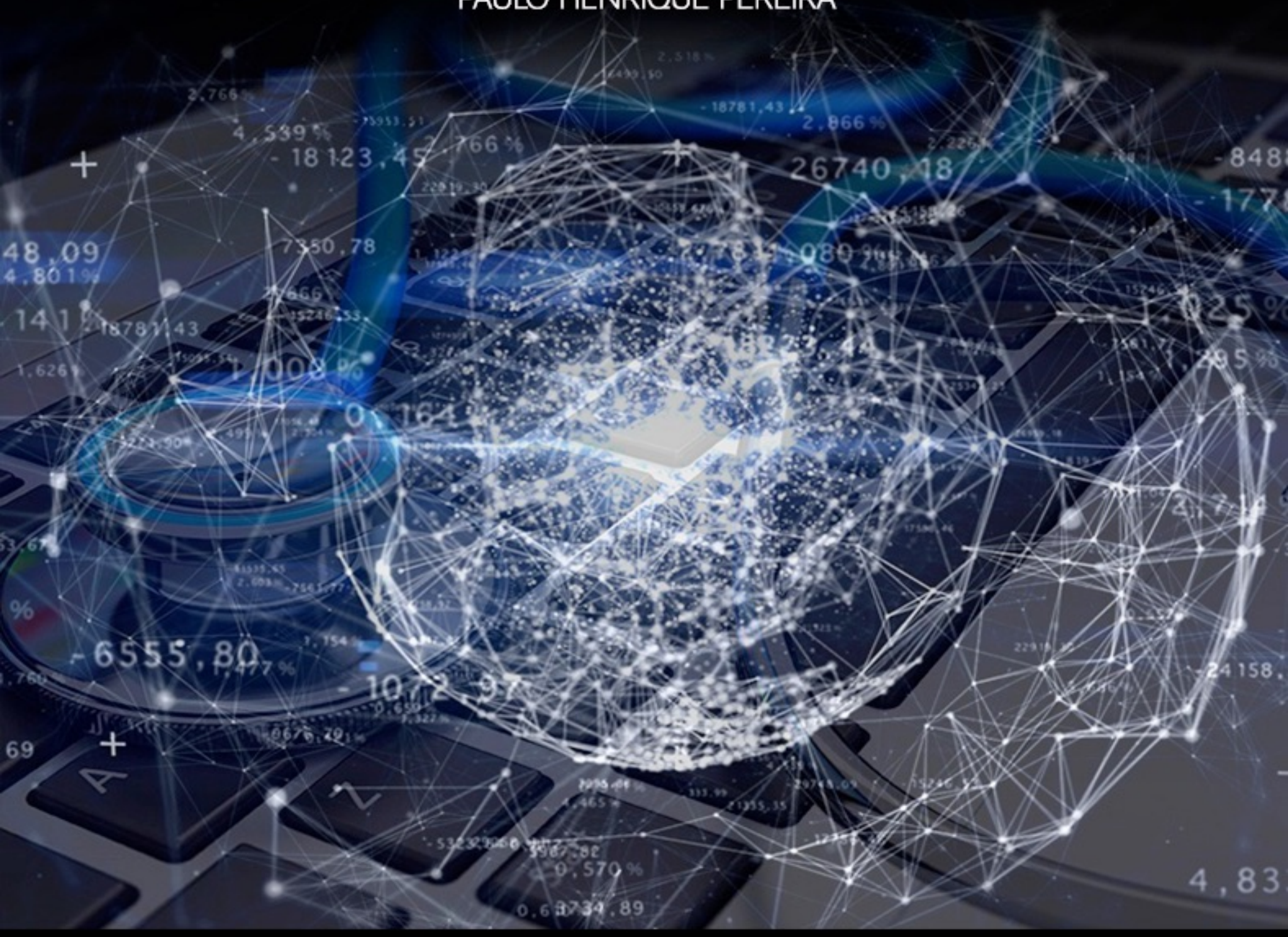
hakin9

WORKSHOPS

MALWARE ANALYSIS USING VOLATILITY

PAULO HENRIQUE PEREIRA



+ EXTRA MATERIALS!

VOL.11 NO.10

HAKIN9

Dear readers,

We would like to proudly present to you the newest Hakin9 workshop issue. In this eBook you will find materials presented in the course "Malware Analysis with Volatility". Volatility introduced people to the power of analyzing the runtime state of a system using the data found in volatile storage (RAM). It also provided a cross-platform, modular, and extensible platform to encourage further work into this exciting area of research. Another major goal of the project was to encourage the collaboration, innovation, and accessibility to knowledge that had been common within the offensive software communities.

Note: Some of the materials, like videos and exercises, are not presented in this issue. If you would like to gain access to all the materials, you have to buy the course.

To make this issue more interesting we decided to add some extra materials. ***Finding Advanced Malware Using Volatility*** by Monnappa KA, that will help you understand how this tool works. In this case scenario you will learn how to detect advanced malware and understand memory forensics. Next two articles were written by Dr. Paulo Henrique Pereira (who is also an instructor of Malware Analysis with Volatility and Live Analysis with Rekall). His first article is about Redline - a tool can collect and analyze data with some scripts. If you want to learn more about that, don't forget to read ***Practical Live Analysis and Auditing Using Redline IOC Models.***

Using n1n3 to simulate an evasive "fileless" malware is a second article written by not only Dr. Pereira, but also by Thiago Geronimo Ferreira, Rubens Louro Vieira, and Renato Basante Borbolla. This article is part of research called Forensics Malware with the use of reverse engineering and is still in progress at the University Nove de Julho (Uninove, Brazil).

The main aim of this eBook is to present our reading materials from our online courses to a wider range of readers. We hope we can meet your expectations. We would also want to thank you for all your support. We appreciate it alot. If you like this publication you can share it and tell your friends about it! Every comment means alot to us. Special thanks to the Proofreader who helped with this issue.

Enjoy your reading,

Hakin9 Magazine's

Editorial Team

TABLE OF CONTENTS

MALWARE ANALYSIS WITH VOLATILITY

MALWARE ANALYSIS WITH VOLATILITY

COURSE MATERIALS

Dr. Paulo Henrique Pereira

SUMMARY

- Setting up a laboratory to do the exercises!
- The setup for Windows and Volatility in the forensic lab.
- The setup for Linux and Volatility in the forensic lab.
- Syntax of commands in Volatility
- References
- Extra materials

Your instructor:

Paulo Henrique Pereira, PhD

Born in São Paulo, Brazil. He has a PhD in the area of analytical induction. Researcher at the University Nove de Julho (UNINOVE) in the area of forensics and security (penetration testing). Works with forensic analysis and reverse engineering of malware. In his spare time, he splits his time between the practice of fly fishing in the rivers that cut through the mountains and programming languages C and Python.

A BRIEF NOTE ABOUT THE IMAGES USED IN THIS COURSE

Volatility is free software. In this course, we do not use any image of Volatility from the Foundation and its forensic training courses. We use our own forensic images created in our virtual machines (or machines that we use in the real world for analysis).

ON THE SOURCES USED IN THIS TEXT

In this course we use a simple notation of commands and regular text:

- **`This is a command.`**
- `This is a plugin name.`
- This is a normal text.
- *This is a text that highlights the importance of a topic.*

BASIC INFORMATION ABOUT VOLATILITY

WHAT IS VOLATILITY?

Volatility is free software developed for forensic analysis purposes by the team Volatility Foundation. Its main proposal *is memory analysis*.

Analysis of RAM is a forensic technique known as *live analysis*. This basically means that we deal with data types that have *a high level of volatility*, i.e., the data can be lost when the machine is turned off.

We can summarize the procedure of live analysis as a process to capture the data that will be lost after the machine is turned off. This is not the case with the user's password, which is a data type that continues to exist even if the machine has been turned off. The settings that a user makes on a Windows machine, for example, such as themes, colors, etc., are types of data with very low volatility.

When we use our computer for a simple task, such as editing a text, we generate a chain of requests that will be mediated by communication between the operating system interface and the machine's hardware.

So when we create a text file, we run a program for this purpose, the text editor. This text editor is called by the user and when it starts working on your computer screen, it is a created process. *Each process has a number and a memory address.*

Suppose a user turns on the computer to do a job, turns it off and then returns later to finish this activity. What happens is that the processes triggered in this second operation will not necessarily be the same as those created when he turned on the computer the first time.

Volatility includes over 200 plugins designed to analyze various features found in RAM. It's a powerful tool for *forensic analysis of malware*. In other words, Volatility can be used to investigate:

- Which libraries have been infected by malware,
- Which services that were running on the system have been compromised by malware,
- If there are any injections of malicious code into a DLL,
- If connections were made with the internet service,
- If the firewall was disabled by malware, and more.

WHAT VOLATILITY IS NOT?

Volatility *is not* an anti-virus software, so its main scope is the forensic exploration of the infection vector of malware, detailing the flow of this infection. Although the framework is able to identify structures for evidence of an infection caused by a malware, Volatility is not intended to clean an infected machine. Its main feature is to make available plugins that can analyze an image of everything that was running in the memory of a machine and that was saved in a file format. In addition, Volatility is not designed to capture an image.

WHAT VOLATILITY DOES NOT INTEND TO DO?

For example, suppose we have a machine and there is a suspicion that it is infected by malware. The same machine was being used in an office. So we can capture all the processes, services and requests made by the user from login on this workstation until the moment that the suspicion was aroused and we need to capture an image of memory. For this purpose, use stand out software, such as DumpIt, Memoryze, FTK Imager and others tools – free edition or paid edition (see [3] for more details).

Below is an example of Belkasoftware Live RAM Capturer (see [4]) capturing the memory of a Dell Vostro machine 360 running Windows 10. The file with the image memory is saved in a directory, in this case:

```
C:\Users\paulo_000\Contacts\Desktop\Uninove\RamCapturer64
```

Figures 1 and 2 below show the output of the RAM Capturer. For the machine mentioned above, this procedure took less than two minutes.

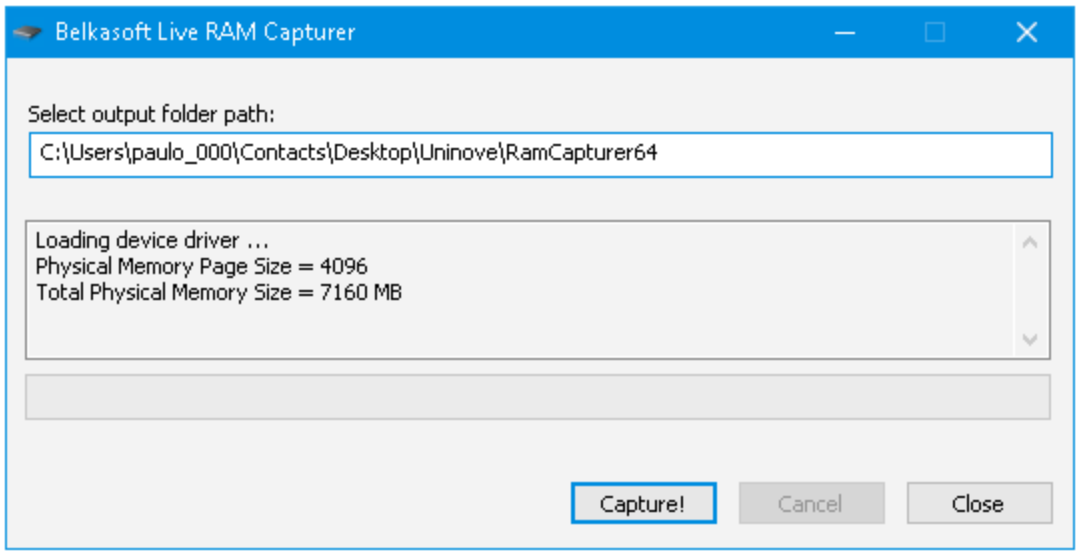


Figure 1: Belkasoft Live RAM Captures initial output.

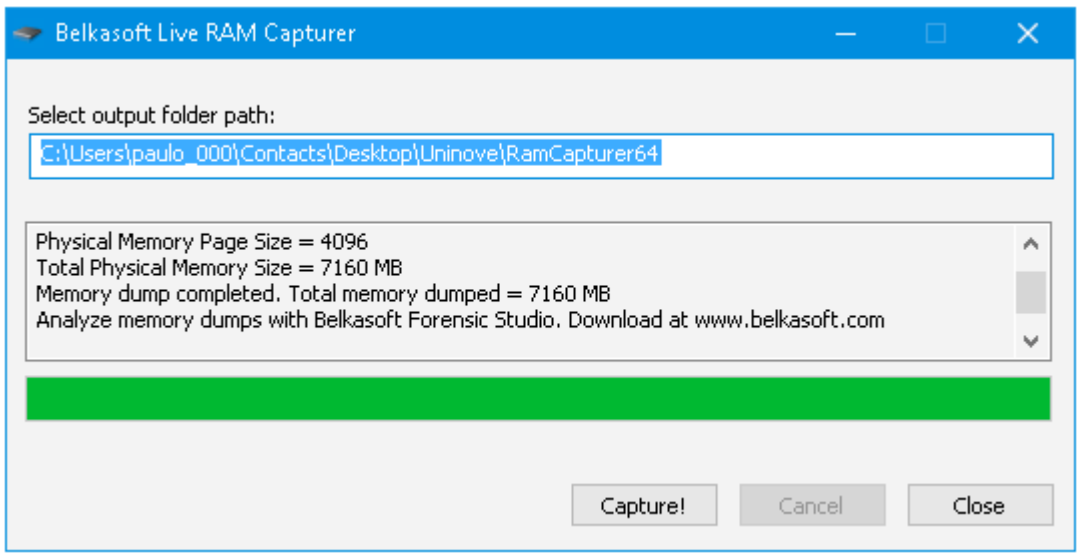


Figure 2: Belkasoft Live RAM Captures final output.

What interests us in terms of Volatility is the file that the Capturer created **20160603.mem**. The machine was running. Note that the file has a bit size of less than 7 GB. This is about the machine's memory capacity.

In this case, Volatility has basic support for this type of image because, as was said above, it was extracted from a machine running the W10 64-bit (until that moment). For earlier versions of Windows, we found broader support.

WHAT IS THE CURRENT VERSION OF VOLATILITY?

The current version of Volatility (at the time this Pre Course is being written) is the 2.5 Standalone Executables for Windows, Mac OSX and Linux released in October 2015. According to the description that we find on the Volatility site, this release brings (see [1] below for reference and credit):

WINDOWS	MAC OSX	LINUX
Added profiles for Windows 8.1 Update 1	Added profiles and support for Mac 10.10 Yosemite and 10.11 El Capitan	Added support for Linux kernels up to 4.2.3
Added basic support for Windows 10	New plugin to print and extract compressed swap data (`mac_compressed_swap`)	New plugin to print Linux dynamic environment variables (`linux_dynamic_env`)
New plugin to print AmCache information from the registry (`amcache`)	New plugin to automatically detect Mac OS X profiles (`mac_get_profile`)	New plugin to print the current working directory of processes (`linux_getcwd`)
New plugin to dump registry files to disk (`dumpregistry`)	New plugin(s) to report Kauth scopes and listeners (`mac_list_kauth_scopes | listeners`)	New plugin to carve for network connection structures (`linux_netscan`)
New plugin to detect hidden/unlinked service record structures (`servicediff`)	New plugin to identify applications with promiscuous sockets (`mac_list_raw`)	Speed improvements to various plugins
New plugin to print the shutdown time from the registry (`shutdowntime`)	New plugin to find hidden threads (`mac_orphan_threads`)	Improve handling of mprotect() Linux memory regions.
New plugin to print editbox controls from the GUI subsystem (`editbox`)	New plugin to print process environment variables (`mac_psenv`)	
Malfind plugin detects injected code with erased PE headers	New plugin to print basic and complex thread data (`mac_threads, mac_threads_simple`).	

Imagecopy and raw2dmp can display the number of bytes copied or converted		
Fix an issue with the memmap and memdump offsets being inconsistent		
Update the well known SIDs reported by the getsids plugin		
Add an optional --max-size parameter to yarascan, dump_maps, etc		
Fix an issue translating strings in PAE and x64 images		
Add options to yarascan for case-insensitive search		
Add options to yarascan to scan process and kernel memory at once		

See http://www.volatilityfoundation.org/25

WHAT MATERIALS WILL BE USED IN THIS COURSE?

- The material of this course is original. Some trainer machines are used to capture memory.
- The images of infected machines are shared and the sites are cited in the references.
- The Volatility Foundation has material that will *not be used in this course* in respect to the use rights.

WHAT PLATFORMS CAN BE ANALYZED BY VOLATILITY?

Volatility can be used in Windows (standalone version 2.5 covers the basics of Windows 10, when this Pre Course is being written), Linux, Mac OSX and Android. As the Volatility project site mentions, the software brings support for operating systems such as (see [1] below):

- 64-bit Windows Server 2012 and 2012 R2
- 32- and 64-bit Windows 10 (initial/basic support)
- 32- and 64-bit Windows 8, 8.1, and 8.1 Update 1
- 32- and 64-bit Windows 7 (all service packs)
- 32- and 64-bit Windows Server 2008 (all service packs)
- 64-bit Windows Server 2008 R2 (all service packs)
- 32- and 64-bit Windows Vista (all service packs)
- 32- and 64-bit Windows Server 2003 (all service packs)
- 32- and 64-bit Windows XP (SP2 and SP3)
- 32- and 64-bit Linux kernels from 2.6.11 to 4.2.3
- 32-bit 10.5.x Leopard (the only 64-bit 10.5 is Server, which isn't supported)
- 32- and 64-bit 10.6.x Snow Leopard
- 32- and 64-bit 10.7.x Lion
- 64-bit 10.8.x Mountain Lion (there is no 32-bit version)
- 64-bit 10.9.x Mavericks (there is no 32-bit version)
- 64-bit 10.10.x Yosemite (there is no 32-bit version)
- 64-bit 10.11.x El Capitan (there is no 32-bit version)

WHAT MEMORY FORMATS CAN BE ANALYZED BY VOLATILITY?

The RAM image formats are different and depend on the software that is used for this capture. Volatility can analyze the following formats (see [1] below):

- Raw/Padded Physical Memory
- Firewire (IEEE 1394)
- Expert Witness (EWF)
- 32- and 64-bit Windows Crash Dump
- 32- and 64-bit Windows Hibernation (from Windows 7 or earlier)
- 32- and 64-bit MachO files
- Virtualbox Core Dumps
- VMware Saved State (.vmss) and Snapshot (.vmsn)
- HPAK Format (FastDump)
- QEMU memory dumps

WHAT SHOULD I INSTALL ON THE MACHINE TO USE VOLATILITY?

On a Windows machine, you do not have to install anything. Just download the latest version of Standalone Volatility. You will use the Command Prompt (CMD). Or, if you prefer, you can use a pen drive.

On a machine running a Linux system, especially the forensic or penetration testing suites, Volatility is already installed. Unless you want to use another Linux distribution (Ubuntu, for example) and download Volatility.

SETTING UP A LABORATORY TO DO THE EXERCISES!

The Windows family requires a license to use, or you can use a trial version if applicable. Many students have access to a license in their respective universities. If this is your case, enjoy! Download an ISO and install it on a virtualization tool. I will use VirtualBox, but we could use VMware without problems.

The important point is that *you should virtualize your lab*. If this is not possible, you can use your laptop or your desktop to perform the exercises. We will deal with images of infected machines and not with malware in executables files. The more you advance in the forensic area will end up dealing with reverse engineering malware. In this case, virtualization is indispensable. So why not start now?

We will show below a basic laboratory that meets those wishing to use Windows or Linux.

Figure 3: A proposal for virtual machines on your forensics lab.

THE SETUP FOR WINDOWS AND VOLATILITY IN THE FORENSIC LAB

Below we show a configuration that is often used for malware analysis in Windows environment using Volatility. Remember, though, that this is not a universal rule. We can change this form of preparation and location of forensic analysis files.

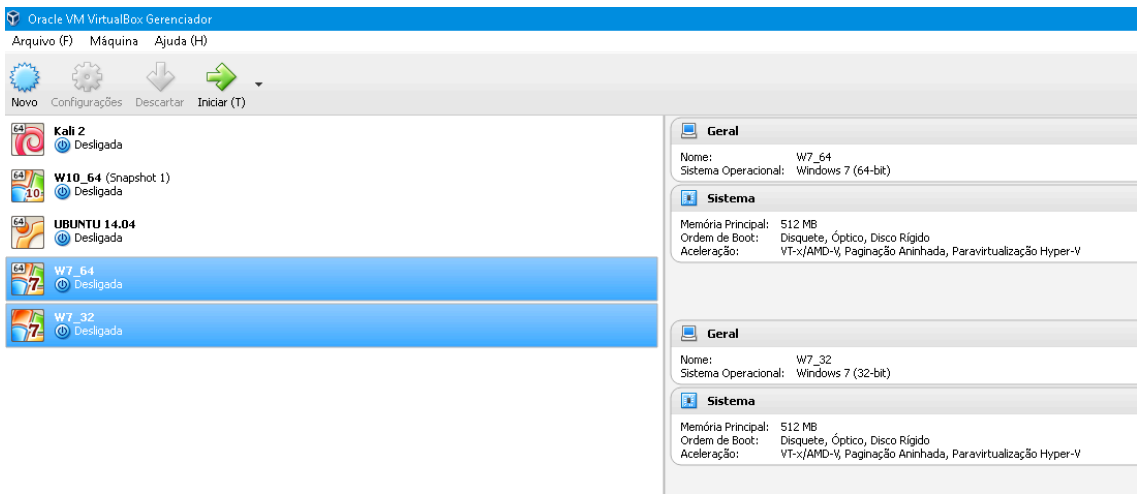


Figure 4: Basic configurations for Windows virtual machines

We installed two machines Windows 7, a 32-bit and one 64 bit (W7_32 and W7_64). In this lab, we set up the two virtual machines with 2048 MB of RAM and 25 GB disk space. These options were chosen by VirtualBox and we chose not to change, however changing is possible if you wish. Of course, you do not need to install both Windows machines. We just chose to expand their possibilities of choices. However, if we were to analyze malware through reverse engineering, we would need the two settings.

Using W7_32 machine to illustrate the next steps (we would give the same steps with the W7_64 machine, install the Live RAM Capturer on that machine. This gives us a snapshot 20160603.mem of the memory of our virtual machine for analysis. We chose to change the file name to W7_32 .mem (you do not need change the name).

Figure 5: Memory image capture from a Windows machine 7 32-bit.

Now we put the W7_32 file in Drive C. Let's download Volatility 2.5, unzip the file and also put it on Drive C (this greatly facilitates our work, believe me!).

The unzipped folder contains six files: AUTHORS, CREDITS, LEGAL, LICENSE, README and **volatility-2.5.standalone**. We are interested in the latter file only. But its name is too big to keep typing, which makes our work boring. So let's rename this file "vol" (without quotes).

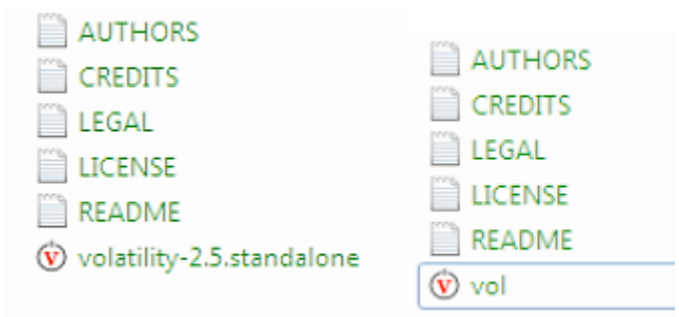


Figure 6a-6b showing the change in the file name.

After these steps you should put both files (vol and W7_32.mem) in the same location (Drive C, for example).

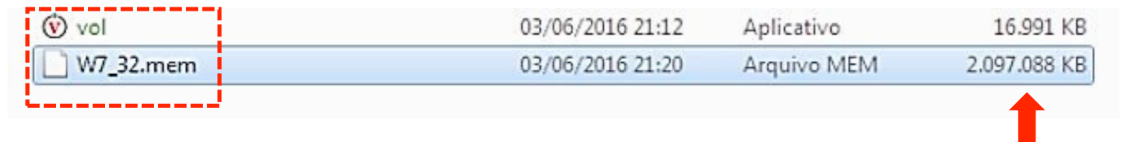


Figure 7: The two files in the same place (Drive C).

Note that the image we are using (W7_32) has the size that has been designated by VirtualBox: 2 GB or a little more than that. Thus, we must always bear in mind that the size of the image reflects the RAM capacity that was intended for your machine (virtual or physical). Depending on the image size and the processing power of a computer that is being used for its forensic laboratory, Volatility may take longer to show a response to the expert, especially when we need to use the `imageinfo` plugins or `kdbgscan` to find the profile operating system that captured the image memory.

Once you have performed these steps, your lab is ready if you choose to use Windows environment.

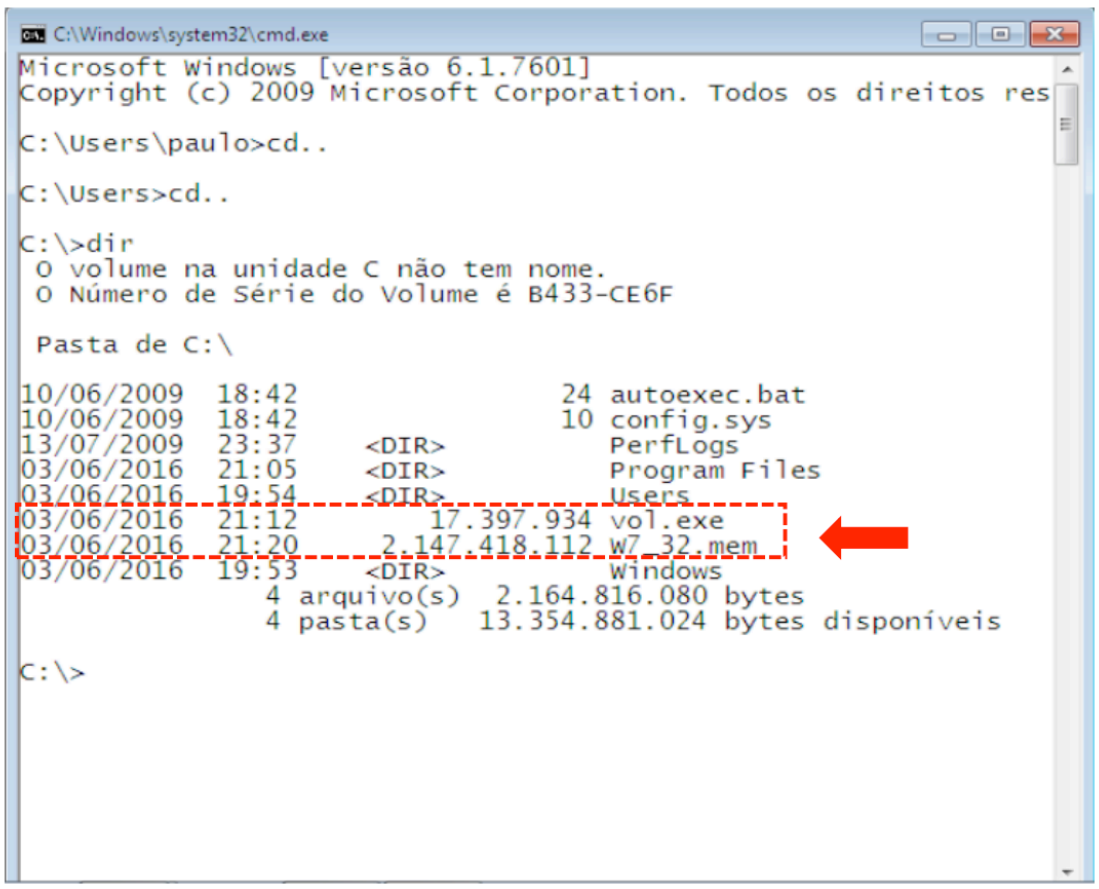


Figure 8: Volatility standalone and image memory placed in the same place to facilitate the work of the forensic analyst.

Whereas the forensic expert's time is made up of very lot of work on a daily basis, the optimization of this time can be achieved when the expert makes it more organized. In the case of Volatility, if we place the image that serves as evidence in a directory that is in a hierarchically below the folder where Volatility is, we must always remember that in an investigation using the command line, this becomes tedious and error-prone.

THE SETUP FOR LINUX AND VOLATILITY IN THE FORENSIC LAB

If you want to work with a Linux platform, in a perspective of information security and computer forensics area we suggest the following options (which we use in our day-to-day):

- Kali Linux 2
- Using as a base UBUNTU 14 can work on a single platform and we use it together with Sans Sift Workstation 3 and REMnux.

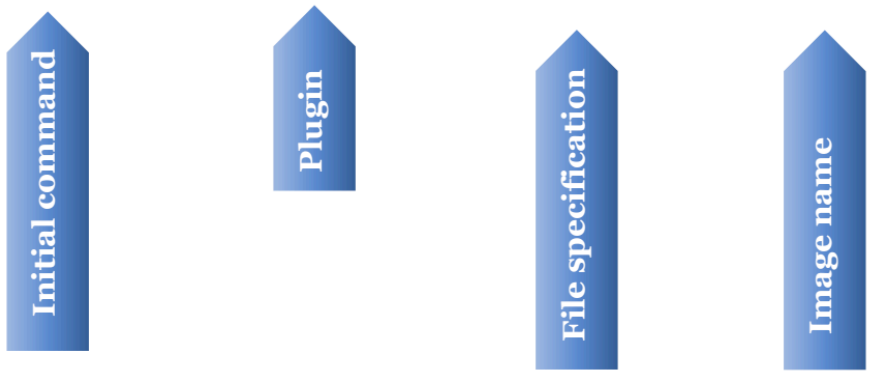
We note that in Kali Linux 2, Volatility is part of the suite and is located in the Forensics directory.

SYNTAX OF COMMANDS IN VOLATILITY

To begin using the Volatility basic commands, follow a structured syntax. On the Windows platform the syntax is as follows:

SYNTAX FOR CORRECT USE OF VOLATILITY (WINDOWS)

```
vol.exe [imageinfo] [-f ] [image.mem]
```

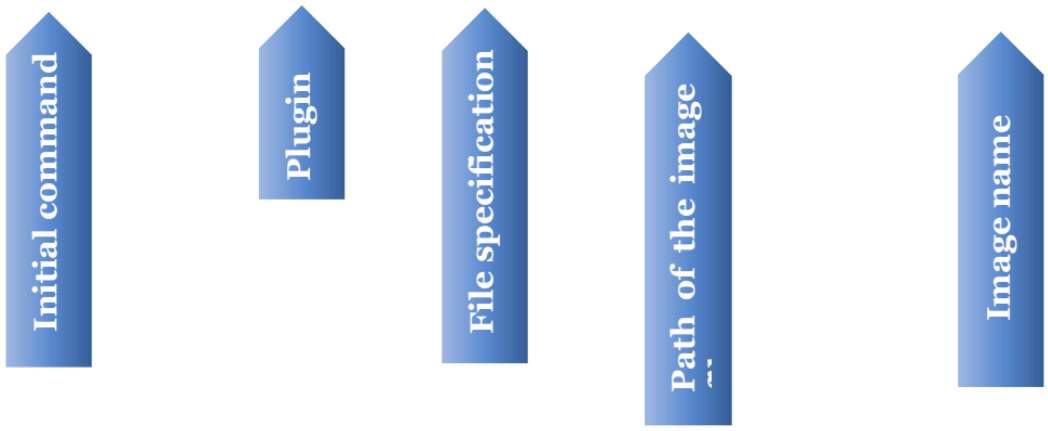


Below is an example:

```
Prompt de Comando
Microsoft Windows [versão 10.0.10586]
(c) 2015 Microsoft Corporation. Todos os direitos reservados.

C:\Users\paulo_000>cd..

C:\Users>cd..

C:\>vol.exe imageinfo -f memdump1.mem
```

SYNTAX FOR CORRECT USE OF VOLATILITY (KALI LINUX 2016.1)

```
volatility imageinfo -f /root/forensics/image.mem
```



Here's an example below:

```
root@kali:~# volatility imageinfo -f /root/forensics/W7_32_512.mem
```

You can also use the command as follows:

```
root@kali:~# volatility -f /root/forensics/W7_32_512.mem imageinfo
```

The `imageinfo` plugin tries to identify the operating system profile and the service pack to the case of the Windows platform. The **`-f`** specification is a reference to a file that will be analyzed. If the image is stored in directories, you must specify the **`path`** to Volatility to find the image. In the example above, our image (W7_32_512.mem) is stored in the folder called forensics, which is within the root directory.

Again we emphasize that the forensic expert can leave the image in the memo Volatility directory or if you prefer, you can create an environment variable (but this is left to the preference of each).

REFERENCES

BOOKS

- LIGH, Michel. H.; CASE, A.; LEVY, J.; WALTERS, A. *The Art of Memory Forensics*. Wiley: Indianapolis, 2014.
- Malin, C. & Casey, E. & Aquilina, J. . *Malware Forensics Field Guide for Windows Systems*. Boston: Syngress, 2012.

SITES

1. http://www.volatilityfoundation.org/#!25/c1f29
2. http://volatilityfoundation.github.io/volatility/annotated.html
3. http://forensicswiki.org/wiki/Tools:Memory_Imaging
4. http://belkasoft.com/ram/download
5. http://www.memoryanalysis.net/#!amf/cmg5
6. https://code.google.com/archive/p/volatility/wikis/VolatilityUsage22.wiki
7. https://github.com/volatilityfoundation/**volatility**/wiki/Memory-Samples
8. http://www.dfrws.org/

EXTRA MATERIALS

FINDING ADVANCED MALWARE USING VOLATILITY

BY MONNAPPA KA

PRACTICAL LIVE ANALYSIS AND AUDITING USING REDLINE IOC MODELS

BY PAULO HENRIQUE PEREIRA

USING NINJA TO SIMULATE AN EVASIVE "FILELESS"

BY PAULO HENRIQUE PEREIRA, THIAGO GERONIMO FERREIRA, RUBENS LOURO VIEIRA, RENATO BASANTE BORBOLLA

FINDING ADVANCED MALWARE USING

by Monnappa KA

WHAT YOU SHOULD KNOW

- Basic understanding of malware
- Knowledge of operating system processes
- Understanding of Windows Internals

WHAT YOU WILL LEARN

- Performing memory forensics
- Tools and techniques to detect advanced malware using Memory forensics
- Volatility usage

Memory Forensics is the analysis of the memory image taken from the running computer. Memory forensics plays an important role in investigations and incident response. It can help in extracting forensics artifacts from a computer's memory like running process, network connections, loaded modules etc. It can also help in unpacking, Rootkit detection and reverse engineering. When an organization is a victim of advanced malware infection, a quick response action is required to identify the indicators associated with that malware to remediate, establish better security controls and to prevent future ones from occurring. In this article you will learn to detect advance malware infection in memory using a technique called "Memory Forensics" and you will also learn to use Memory Forensic Toolkits such as Volatility to detect advanced malware with a real case scenario.

STEPS IN MEMORY FORENSICS

Below is the list of steps involved in memory forensics:

a) **Memory Acquisition -** This step involves dumping the memory of the target machine. On the physical machine you can use tools like *Win32dd/Win64dd, Memoryze, DumpIt, FastDump*. Whereas on the virtual machine, acquiring the memory image is easy, you can do it by suspending the VM and grabbing the ".vmem" file.

b) **Memory Analysis -** once a memory image is acquired, the next step is to analyze the grabbed memory dump for forensic artifacts, tools like *Volatility* and others like Memoryze can be used to analyze the memory

VOLATILITY QUICK OVERVIEW

Volatility is an advanced memory forensic framework written in python. Once the memory image has been acquired Volatility framework can be used to perform memory forensics on the acquired memory image. Volatility can be installed on multiple operating systems (Windows, Linux, Mac OS X), Installation details of volatility can be found at *http://code.google.com/p/volatility/wiki/FullInstallation*

VOLATILITY SYNTAX

- Using -h or --help option will display help options and list of a available plugins

Example:

```
python vol.py -h
```

- Use -f <filename> and --profile to indicate the memory dump you are analyzing

Example:

```
python vol.py -f mem.dmp --profile=WinXPSP3x86
```

- To know the --profile info use below command:

Example:

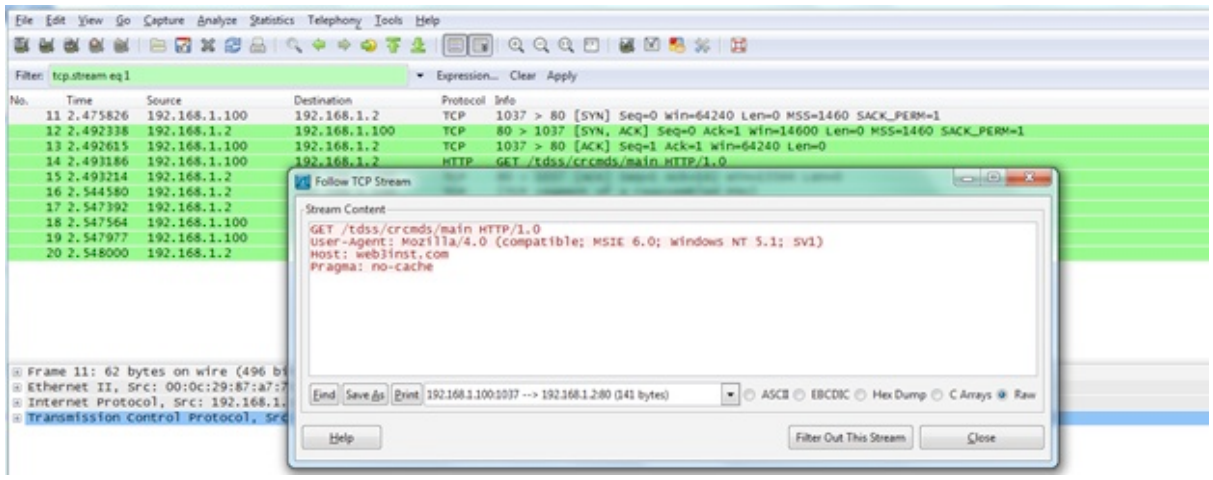
```
python vol.py -f mem.dmp imageinfo
```

DEMO

In order to understand memory forensics and the steps involved. Let's look at a case scenario, our analysis and flow will be based on the below case scenario.

CASE SCENARIO

Your security device alerts on malicious http connection to the domain "web3inst.com" which resolves to 192.168.1.2, communication is detected from a source ip 192.168.1.100 (as shown in the below screenshot).you are asked to investigate and perform memory forensics on the machine 192.168.1.100



MEMORY ACQUISITION

To start with, acquire the memory image from 192.168.1.100, using memory acquisition tools. For the sake of demo, the memory dump file is named as "infected.vmem".

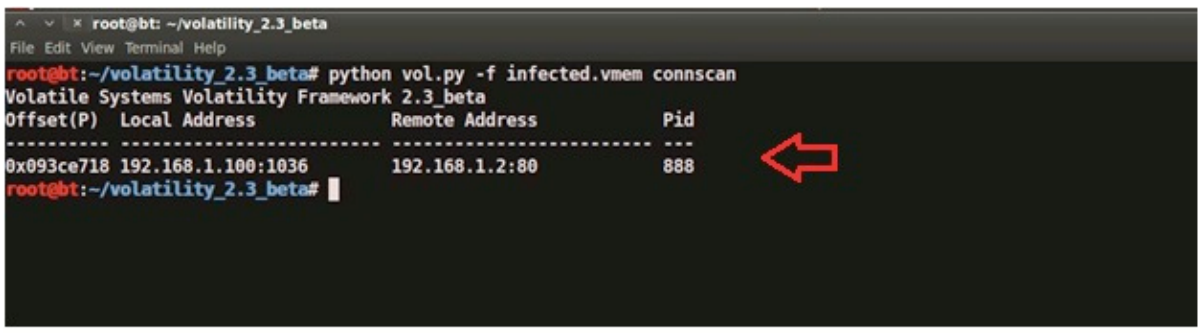
ANALYSIS

Now that we have acquired "infected.vmem", let's start our analysis using Volatility advanced memory analysis framework

STEP 1: START WITH WHAT YOU KNOW

We know from the security device alert that the host was making an http connection to *web3inst.com (192.168.1.2)*. So let's look at the network connections.

Volatility's connscan module, shows connection to the malicious ip made by process (with pid 888)



STEP 2: INFO ABOUT WEB3INST.COM

Google search shows this domain(web3inst.com) is known to be associated with malware, probably "Rustock or TDSS Rootkit". This indicates that source ip 192.168.1.100 could be infected by any of these malwares, we need to confirm that with further analysis.

STEP 3: WHAT IS PID 888?

Since the network connection to the ip 192.168.1.2 was made by pid 888, we need to determine which process is associated with pid 888. "psscan" shows pid 888 belongs to svchost.exe.

```
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem psscan
Volatile Systems Volatility Framework 2.3_beta
Offset(P)  Name                PID   PPID PDB        Time created                   Time exited
---------- ---------------- ------ ------ ---------- ------------------------------ ------------------------------
0x0919fa70 wmiprvse.exe        780    888 0x0ec80240 2012-08-15 17:08:33 UTC+0000
0x09300020 alg.exe            1568    700 0x0ec80180 2012-08-15 17:08:34 UTC+0000
0x0931cda0 winlogon.exe        656    376 0x0ec80060 2012-08-15 17:08:22 UTC+0000
0x093db348 VMwareTray.exe     1744    560 0x0ec80260 2012-08-15 17:08:34 UTC+0000
0x093e72c0 VMwareUser.exe     1752    560 0x0ec80280 2012-08-15 17:08:34 UTC+0000
0x09418be0 wuauclt.exe        1596   1052 0x0ec802a0 2012-10-07 12:46:56 UTC+0000
0x0941ca20 tdl3.exe           1468   1752 0x0ec802c0 2012-10-07 12:46:57 UTC+0000   2012-10-07 12:46:57 UTC+0000
0x09431da0 VMUpgradeHelper     224    700 0x0ec801e0 2012-08-15 17:08:33 UTC+0000
0x09439b28 vmtoolsd.exe       1976    700 0x0ec801c0 2012-08-15 17:08:30 UTC+0000
0x0943c778 msiexec.exe        1236    700 0x0ec802e0 2012-10-07 12:46:57 UTC+0000
0x09445af0 explorer.exe        560    460 0x0ec80220 2012-08-15 17:08:33 UTC+0000
0x09446da0 spoolsv.exe        1388    700 0x0ec801a0 2012-08-15 17:08:24 UTC+0000
0x09457520 services.exe        700    656 0x0ec80080 2012-08-15 17:08:22 UTC+0000
0x094d7020 svchost.exe        1128    700 0x0ec80160 2012-08-15 17:08:22 UTC+0000
0x094dada0 svchost.exe        1052    700 0x0ec80120 2012-08-15 17:08:22 UTC+0000
0x094df530 svchost.exe         968    700 0x0ec80100 2012-08-15 17:08:22 UTC+0000
0x094e0aa0 svchost.exe        1096    700 0x0ec80140 2012-08-15 17:08:22 UTC+0000
0x094e6878 vmacthlp.exe        868    700 0x0ec800c0 2012-08-15 17:08:22 UTC+0000
0x094ea5d8 svchost.exe         888    700 0x0ec800e0 2012-08-15 17:08:22 UTC+0000
0x094f18e8 csrss.exe           632    376 0x0ec80040 2012-08-15 17:08:21 UTC+0000
0x095f98e8 smss.exe            376      4 0x0ec80020 2012-08-15 17:08:20 UTC+0000
```

STEP 4: YARA SCAN

Running the YARA scan on the memory dump for the string "web3inst" confirms that this domain (web3inst.com) is present in the address space of svchost.exe (pid 888). This confirms that svchost.exe was making connections to the malicious domain "web3inst.com".

```
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem yarascan -Y "web3inst"
Volatile Systems Volatility Framework 2.3_beta
Rule: r1
Owner: Process svchost.exe Pid 888
0x1000470b  77 65 62 33 69 6e 73 74 2e 63 6f 6d 2f 74 64 73   web3inst.com/tds
0x1000471b  73 2f 63 72 63 6d 64 73 2f 6d 61 69 6e 00 00 00   s/crcmds/main...
0x1000472b  00 68 74 74 70 3a 2f 2f 77 65 62 34 69 6e 73 74   .http://web4inst
0x1000473b  2e 63 6f 6d 2f 74 64 73 73 2f 63 72 63 6d 64 73   .com/tdss/crcmds
```

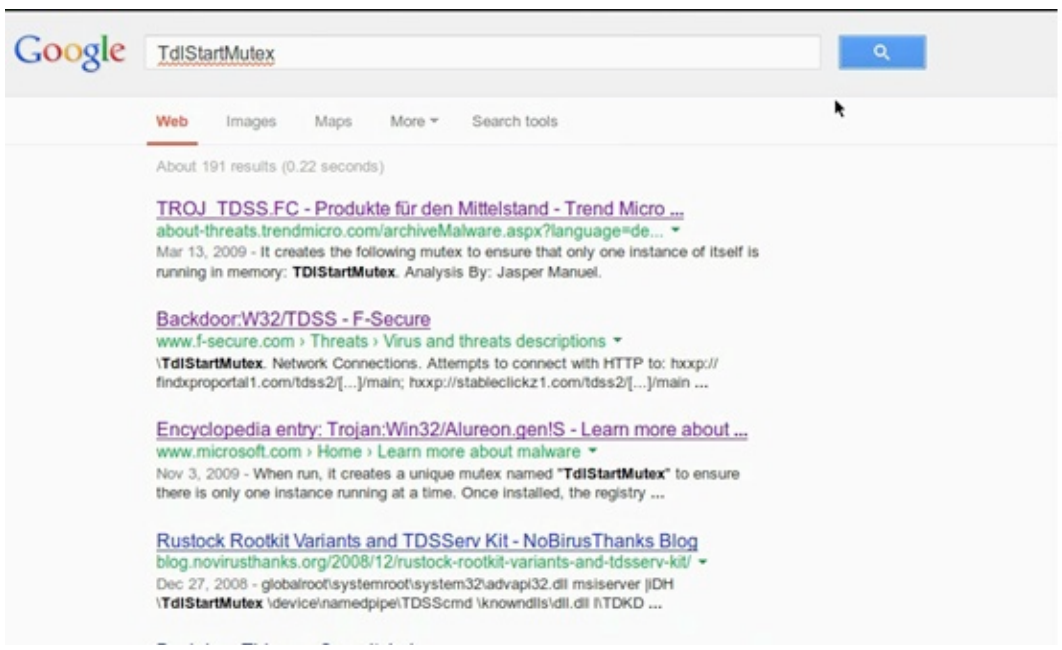
STEP 5: SUSPICIOUS MUTEX IN SVCHOST.EXE

Now we know that svchost.exe process (pid 888) was making connections to the domain "web3inst.com", lets focus on this process. Checking for the mutex created by svchost.exe shows a suspicious mutex "TdlStartMutex".

```
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem handles -p 888 -t Mutant
Volatile Systems Volatility Framework 2.3_beta
Offset(V)     Pid     Handle     Access Type            Details
---------- ------ ---------- ---------- --------------- -------
0x88fdda88    888       0x24   0x1f0001 Mutant          SHIMLIB_LOG_MUTEX
0x88fd16f8    888       0x15c  0x1f0001 Mutant          {A3BD3259-3E4F-428a-84C8-F0463A9D3EB5}
0x89258020    888       0x164  0x1f0001 Mutant
0x8921f838    888       0x1e0  0x1f0001 Mutant
0x89534fa0    888       0x1ec  0x120001 Mutant          ShimCacheMutex
0x890e95f8    888       0x1f8  0x1f0001 Mutant
0x8921f7f8    888       0x200  0x1f0001 Mutant
0x8921f788    888       0x208  0x1f0001 Mutant
0x88f8c720    888       0x220  0x1f0001 Mutant          746bbf3569adEncrypt
0x89219ce8    888       0x240  0x1f0001 Mutant
0x88f94340    888       0x28c  0x1f0001 Mutant
0x895324a8    888       0x34c  0x1f0001 Mutant          TdlStartMutex
0x890ea2b0    888       0x3d8  0x120001 Mutant          DBWinMutex
0x88fc9648    888       0x3f4  0x100000 Mutant          _!MSFTHISTORY!_
0x894968d8    888       0x408  0x1f0001 Mutant          c:!windows!system32!config!systemprofile!local settings!temporary internet files!co
ent.ie5!
0x894abda8    888       0x414  0x1f0001 Mutant          c:!windows!system32!config!systemprofile!cookies!
0x894ab790    888       0x420  0x1f0001 Mutant          c:!windows!system32!config!systemprofile!local settings!history!history.ie5!
0x890f72f0    888       0x430  0x100000 Mutant          WininetStartupMutex
0x891dbd48    888       0x434  0x1f0001 Mutant
0x89249498    888       0x438  0x100000 Mutant          WininetProxyRegistryMutex
0x8923cbd8    888       0x448  0x1f0001 Mutant
0x88fbf800    888       0x454  0x100000 Mutant          RasPbFile
0x891ef860    888       0x4b0  0x1f0001 Mutant          ZonesCounterMutex
0x891df878    888       0x538  0x1f0001 Mutant          ZonesLockedCacheCounterMutex
```

STEP 6: INFO ABOUT THE MUTEX

Google search shows that this suspicious mutex is associated with TDSS rootkit. This indicates that the mutex "TdlStartMutex" is malicious.



STEP 7: FILE HANDLES OF SVCHOST.EXE

Examining file handles in svchost.exe (pid 888) shows handles to two suspicious files (DLL and driver file). As you can see in the below screenshot both these files start with "TDSS".

```
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem handles -p 888 -t File
Volatile Systems Volatility Framework 2.3_beta
Offset(V)     Pid     Handle     Access Type            Details
---------- ------ ---------- ---------- --------------- -------
0x8924d418    888      0x154   0x12019f File            \Device\WMIDataDevice
0x89493d08    888      0x290   0x12019f File            \Device\Termdd
0x890d9db0    888      0x298   0x12019f File            \Device\Termdd
0x892cc678    888      0x2d0   0x12019f File            \Device\NamedPipe\Ctx_WinStation_API_service
0x893dfae0    888      0x2d4   0x12019f File            \Device\NamedPipe\Ctx_WinStation_API_service
0x891eb458    888      0x2f4   0x12019f File            \Device\Termdd
0x891eb390    888      0x2f8   0x12019f File            \Device\Termdd
0x894962b0    888      0x328   0x12019f File            \Device\WMIDataDevice
0x890fd338    888      0x340   0x100020 File            \Device\HarddiskVolume1\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-Controls_6595b641
44ccf1df_6.0.2600.5512_x-ww_35d4ce83
0x88f9ad98    888      0x348   0x120089 File            \Device\HarddiskVolume1\WINDOWS\system32\TDSSoiqh.dll
0x88f7dbe0    888      0x350   0x120089 File            \Device\HarddiskVolume1\WINDOWS\system32\drivers\TDSSmqxt.sys
0x892bb008    888      0x354      0x187 File            \Device\NamedPipe\TDSScmd
0x89248c68    888      0x35c      0x187 File            \Device\NamedPipe\TDSScmd
0x892189d0    888      0x360      0x187 File            \Device\NamedPipe\TDSScmd
0x89109888    888      0x364      0x187 File            \Device\NamedPipe\TDSScmd
0x8948abd0    888      0x368      0x187 File            \Device\NamedPipe\TDSScmd
```

STEP 8: DETECTING HIDDEN DLL

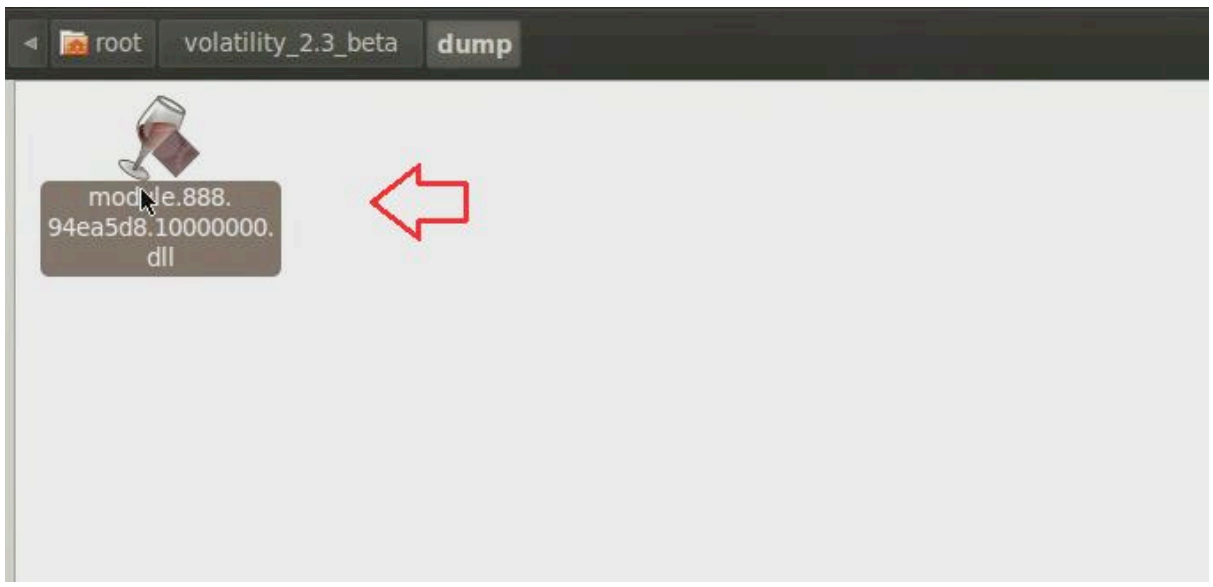
Volatility's dlllist module couldn't find the DLL starting with "TDSS" whereas ldrmodules plugin was able to find it. This confirms that the DLL (TDSSoiqh.dll) was hidden. malware hides the DLL by unlinking from the 3 PEB lists (operating sytem keeps track of the DLL's in these lists).

```
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem dlllist -p 888 | grep -i tdss
Volatile Systems Volatility Framework 2.3_beta
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem ldrmodules -p 888 | grep -i tdss
Volatile Systems Volatility Framework 2.3_beta
     888 svchost.exe          0x10000000 False  False  False \WINDOWS\system32\TDSSoiqh.dll
root@bt:~/volatility_2.3_beta#
```

STEP 9: DUMPING THE HIDDEN DLL

In the previous step hidden DLL was detected. This hidden DLL can be dumped from the memory to disk using Volatility's dlldump module as shown below:

```
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem dlldump -p 888 -b 0x10000000 -D dump
Volatile Systems Volatility Framework 2.3_beta
Process(V) Name                  Module Base Module Name          Result
---------- --------------------- ----------- -------------------- ------
0x892ea5d8 svchost.exe           0x010000000 UNKNOWN              OK: module.888.94ea5d8.10000000.dll
root@bt:~/volatility_2.3_beta#
```



STEP 10: VIRUSTOTAL SUBMISSION OF DUMPED DLL

Submitting the dumped dll to VirusTotal confirms that it is malicious.

GData	Gen:Trojan.Heur.GM.0000610110	20130709
Ikarus	Packed.Win32.Krap	20130709
Jiangmin	✓	20130709
K7AntiVirus	Riskware	20130709
K7GW	Riskware	20130709
Kaspersky	✓	20130709
Kingsoft	Win32.Troj.Undef.(kcloud)	20130708
Malwarebytes	✓	20130709
McAfee	Artemis!3CCE3463DB2E	20130709
McAfee-GW-Edition	Artemis!3CCE3463DB2E	20130709
Microsoft	VirTool:Win32/Obfuscator.DQ	20130709
MicroWorld-eScan	✓	20130709
NANO-Antivirus	Trojan.Win32.Tdss.qfplb	20130709
Norman	✓	20130708
nProtect	✓	20130709
Panda	Generic Worm	20130709
PCTools	Trojan.Gen	20130709

STEP 11: LOOKING FOR OTHER MALICIOUS DLL'S

Looking for the modules in all the processes that start with "TDSS" shows that msiexec.exe process (pid 1236) has reference to a temp file (which is starting with TDSS) which is suspicious.

```
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem ldrmodules | grep -i tdss
Volatile Systems Volatility Framework 2.3_beta
     888 svchost.exe          0x10000000 False  False  False \WINDOWS\system32\TDSSoiqh.dll
    1236 msiexec.exe          0x10000000 True   True   True  \DOCUME~1\ADMINI~1\LOCALS~1\Temp\TDSSf184.tmp
    1236 msiexec.exe          0x77dd0000 True   True   True  \DOCUME~1\ADMINI~1\LOCALS~1\Temp\TDSSf193.tmp
root@bt:~/volatility_2.3_beta#
```

STEP 12: SUSPICIOUS DLL LOADED BY MSIEXEC

Examining the DLL's loaded by the process msiexec (pid 1236) using dlllist module, shows a suspicious dll (dll.dll) loaded by msiexec process.

```
************************************************************************
msiexec.exe pid:   1236
Command line : C:\WINDOWS\system32\msiexec.exe /V
Service Pack 3

Base             Size  LoadCount Path
---------- ---------- ---------- ----
0x01000000    0x16000     0xffff C:\WINDOWS\system32\msiexec.exe
0x7c900000    0xaf000     0xffff C:\WINDOWS\system32\ntdll.dll
0x7c800000    0xf6000     0xffff C:\WINDOWS\system32\kernel32.dll
0x77c10000    0x58000     0xffff C:\WINDOWS\system32\msvcrt.dll
0x77dd0000    0x9b000     0xffff C:\WINDOWS\system32\ADVAPI32.dll
0x77e70000    0x92000     0xffff C:\WINDOWS\system32\RPCRT4.dll
0x77fe0000    0x11000     0xffff C:\WINDOWS\system32\Secur32.dll
0x7e410000    0x91000     0xffff C:\WINDOWS\system32\USER32.dll
0x77f10000    0x49000     0xffff C:\WINDOWS\system32\GDI32.dll
0x774e0000   0x13d000     0xffff C:\WINDOWS\system32\ole32.dll
0x7d1e0000   0x2bc000     0xffff C:\WINDOWS\system32\msi.dll
0x5cb70000    0x26000        0x1 C:\WINDOWS\system32\ShimEng.dll
0x6f880000   0x1ca000        0x1 C:\WINDOWS\AppPatch\AcGenral.DLL
0x76b40000    0x2d000        0x2 C:\WINDOWS\system32\WINMM.dll
0x77120000    0x8b000        0x3 C:\WINDOWS\system32\OLEAUT32.dll
0x77be0000    0x15000        0x1 C:\WINDOWS\system32\MSACM32.dll
0x77c00000     0x8000        0x3 C:\WINDOWS\system32\VERSION.dll
0x7c9c0000   0x817000        0x1 C:\WINDOWS\system32\SHELL32.dll
0x77f60000    0x76000        0x5 C:\WINDOWS\system32\SHLWAPI.dll
0x769c0000    0xb4000        0x1 C:\WINDOWS\system32\USERENV.dll
0x5ad70000    0x38000        0x1 C:\WINDOWS\system32\UxTheme.dll
0x10000000    0x2b000        0x1 C:\WINDOWS\system32\dll.dll
0x76390000    0x1d000        0x1 C:\WINDOWS\system32\IMM32.DLL
0x773d0000   0x103000        0x3 C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-Controls_6595b64144ccf1df_6.0.260
.dll
```

STEP 13: DUMPING DLL AND VT SUBMISSION

Dumping the suspicious DLL (dll.dll) and submitting to VirusTotal confirms that this is associated with TDSS (Alueron) rootkit:

```
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem dlldump -p 1236 -b 0x10000000 -D dump
Volatile Systems Volatility Framework 2.3_beta
Process(V) Name                Module Base Module Name          Result
---------- ------------------- ----------- ------------------- ------
0x8923c778 msiexec.exe         0x010000000 dll.dll             OK: module.1236.943c778.10000000.dll
root@bt:~/volatility_2.3_beta#
```

ClamAV	✓	20130709
Commtouch	✓	20130709
Comodo	✓	20130709
DrWeb	BackDoor.Tdss.30	20130709
Emsisoft	Trojan.Dropper.STN (B)	20130709
eSafe	✓	20130709
ESET-NOD32	✓	20130709
F-Prot	✓	20130709
F-Secure	Trojan.Dropper.STN	20130709
Fortinet	✓	20130709
GData	Trojan.Dropper.STN	20130709
Ikarus	Trojan.Win32.Alureon	20130709
Jiangmin	✓	20130709
K7AntiVirus	✓	20130709
K7GW	✓	20130709
Kaspersky	✓	20130709
Kingsoft	Win32.Troj.TDSS.de.102400	20130708

STEP 14: HIDDEN KERNEL DRIVER

In step 7 we also saw reference to a driver file (starting with "TDSS"). Searching for the driver file using Volatility's modules plugin couldn't find the driver that starts with "TDSS" whereas Volatility's driverscan plugin was able to find it. This confirms that the kernel driver (TDSSserv.sys) was hidden. The below screen-shot also shows that the base address of the driver is "0xb838b000" and the size is "0x11000".

```
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem modules | grep -i tdss
Volatile Systems Volatility Framework 2.3_beta
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem driverscan | grep -i tdss
Volatile Systems Volatility Framework 2.3_beta
0x09732f38    2    0 0xb838b000    0x11000 TDSSserv.sys                    \Driver\TDSSserv.sys
root@bt:~/volatility_2.3_beta#
```

STEP 15: KERNEL CALLBACKS

Examining the callbacks shows the callback (at address starting with 0xb38) set by an unknown driver.

```
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem callbacks
Volatile Systems Volatility Framework 2.3_beta
Type                                Callback   Module              Details
----------------------------------- ---------- ------------------- -------
```

```
IoRegisterShutdownNotification      0xba53fc6a VIDEOPRT.SYS         \Driver\mnmdd
IoRegisterShutdownNotification      0xba53fc6a VIDEOPRT.SYS         \Driver\RDPCDD
IoRegisterShutdownNotification      0xba53fc6a VIDEOPRT.SYS         \Driver\VgaSave
IoRegisterShutdownNotification      0xba53fc6a VIDEOPRT.SYS         \Driver\vmx_svga
IoRegisterShutdownNotification      0xbadb65be Fs_Rec.SYS           \FileSystem\Fs_Rec
IoRegisterShutdownNotification      0xbadb65be Fs_Rec.SYS           \FileSystem\Fs_Rec
IoRegisterShutdownNotification      0xba8b873a MountMgr.sys         \Driver\MountMgr
IoRegisterShutdownNotification      0xba74a2be ftdisk.sys           \Driver\Ftdisk
IoRegisterShutdownNotification      0xba5e78f1 Mup.sys              \FileSystem\Mup
IoRegisterShutdownNotification      0x805cdef4 ntoskrnl.exe         \FileSystem\RAW
IoRegisterShutdownNotification      0x805f5d66 ntoskrnl.exe         \Driver\WMIxWDM
GenericKernelCallback               0xb838e108 UNKNOWN              -
GenericKernelCallback               0xb838d8e9 UNKNOWN              -
GenericKernelCallback               0xbadfeafe CaptureRe...itor.sys -
GenericKernelCallback               0xbadfa7b4 CapturePr...itor.sys -
KeRegisterBugCheckReasonCallback    0xbad74ab8 mssmbios.sys         SMBiosDa
KeRegisterBugCheckReasonCallback    0xbad74a70 mssmbios.sys         SMBiosRe
KeRegisterBugCheckReasonCallback    0xbad74a28 mssmbios.sys         SMBiosDa
KeRegisterBugCheckReasonCallback    0xba51c1be USBPORT.SYS          USBPORT
KeRegisterBugCheckReasonCallback    0xba51c11e USBPORT.SYS          USBPORT
KeRegisterBugCheckReasonCallback    0xba533522 VIDEOPRT.SYS         Videoprt
PsSetLoadImageNotifyRoutine         0xb838e108 UNKNOWN              -
PsSetCreateProcessNotifyRoutine     0xbadfa7b4 CapturePr...itor.sys -
PsSetCreateProcessNotifyRoutine     0xb838d8e9 UNKNOWN              -
CmRegisterCallback                  0xbadfeafe CaptureRe...itor.sys -
root@bt:~/volatility_2.3_beta#
```

STEP 16: EXAMINING THE UNKNOWN KERNEL DRIVER

The below screenshot shows that this unknown driver falls under the address range of TDSSserv.sys. This confirms that unknown driver is "TDSSserv.sys".

```
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem driverscan | grep -i  0xb838
Volatile Systems Volatility Framework 2.3_beta
0x09732f38    2    0 0xb838b000    0x11000 TDSSserv.sys                    \Driver\TDSSserv.sys
root@bt:~/volatility_2.3_beta#
```

STEP 17: KERNEL API HOOKS

Malware hooks the Kernel API and the hook address falls under the address range of TDSSserv.sys (as shown in the below screenshots).

```
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem apihooks -P -Q
Volatile Systems Volatility Framework 2.3_beta
```

```
********************************************************************************
Hook mode: Kernelmode
Hook type: Inline/Trampoline
Victim module: ntoskrnl.exe (0x804d7000 - 0x806cf580)
Function: ntoskrnl.exe!IofCompleteRequest at 0x804ee1b0
Hook address: 0xb838d6bb
Hooking module: <unknown>

Disassembly(0):
0x804ee1b0 ff2504c25480     JMP DWORD [0x8054c204]
0x804ee1b6 cc               INT 3
0x804ee1b7 cc               INT 3
0x804ee1b8 cc               INT 3
0x804ee1b9 cc               INT 3
0x804ee1ba cc               INT 3
0x804ee1bb cc               INT 3
0x804ee1bc 8bff             MOV EDI, EDI
0x804ee1be 55               PUSH EBP
0x804ee1bf 8bec             MOV EBP, ESP
0x804ee1c1 56               PUSH ESI
0x804ee1c2 ff1514774d80     CALL DWORD [0x804d7714]

Disassembly(1):
```

```
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem driverscan | grep -i  0xb838
Volatile Systems Volatility Framework 2.3_beta
0x09732f38    2    0 0xb838b000    0x11000 TDSSserv.sys                \Driver\TDSSserv.sys
root@bt:~/volatility_2.3_beta#
```

STEP 18: DUMPING THE KERNEL DRIVER

Dumping the kernel driver and submitting it to VirusTotal confirms that it is TDSS (Alureon) rootkit.

```
root@bt:~/volatility_2.3_beta# python vol.py -f infected.vmem moddump -b 0xb838b000 -D dump
Volatile Systems Volatility Framework 2.3_beta
Module Base Module Name          Result
---------- -------------------- ------
0x0b838b000 UNKNOWN              OK: driver.b838b000.sys
root@bt:~/volatility_2.3_beta#
```

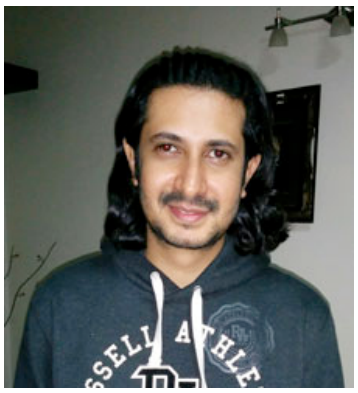
ESET-NOD32	✔	20130709
F-Prot	W32/Trojan3.WZ	20130709
F-Secure	Gen:Rootkit.Heur.du8@diuKQjgi	20130709
Fortinet	W32/TDSS.B!tr	20130709
GData	Gen:Rootkit.Heur.du8@diuKQjgi	20130709
Ikarus	Trojan.Win32.Alureon	20130709
Jiangmin	✔	20130709
K7AntiVirus	Trojan	20130709
K7GW	✔	20130709
Kaspersky	UDS:DangerousObject.Multi.Generic	20130709
Kingsoft	Win32.Troj.Generic.a.(kcloud)	20130708
Malwarebytes	✔	20130709
McAfee	generic!bg.bcg	20130709
McAfee-GW-Edition	generic!bg.bcg	20130709
Microsoft	Trojan:WinNT/Alureon.D	20130709
MicroWorld-eScan	✔	20130709
NANO-Antivirus	Trojan.Win32.ZPACK.zkens	20130709
Norman	TDSSServ.AM	20130708

CONCLUSION

Memory forensics is a powerful investigation technique and with a tool like Volatility it is possible to find advanced malware and its forensic artifacts from the memory which helps in incident response, malware analysis and reverse engineering. As you saw, starting with little information we were able to detect the advanced malware and its components.

Article from eForenisics blog: https://eforensicsmag.com/finding-advanced-malware-using-volatility/

ABOUT THE AUTHOR:



Monnappa K A is based out of Bangalore, India. He has an experience of 7 years in the security domain. He works with Cisco Systems as Information Security Investigator. He is also the member of a security research community SecurityXploded (SX). Besides his job routine he does reasearch on malware analysis and reverse engineering, he has presented on various topics like "Memory Forensics", "Advanced Malware Analysis", "Rootkit Analysis", "Detection and Removal of Malwares" and "Sandbox Analysis" in the Bangalore security community meetings. His article on "Malware Analysis" was also published in the Haking ebook "Malware - From Basic Cleaning To Analyzing"

You can view the video demo's of all his presentations by subscribing to his youtube channel: http://www.youtube.com/user/hackycracky22

or visit his website: https://cysinfo.com