Extracting R2D2 Malware from Memory Using Volatility

Thet Thet Khaing
University of Computer Studies (Yangon)
thetthetkhaing@ucsy.edu.mm

Phyu Sin Nyein
University of Computer Studies (Yangon)
phyusinnyein17@gmail.com

Abstract

There is still a growing interest in memory forensics technology today. There are several Cyber Forensics techniques and tools available to help combat Cyber Crime. Among them, memory forensics refer to the analysis of volatile data in a computer's memory dump. Volatile data is the data stored in RAM on a computer while it is running. Memory forensics can also be triggered the running process of the system such as open connections, recent executed commands in a computer. This paper analyses the memory to detect malware by using volatility tools. This tool is popular volatile memory software analyzer. It can help us to recover useful information stored on the memory of the computer. This extracting scheme focuses on applied operations of memory forensics in Kali Linux machine to detect Trojan malware R2D2 attacks. This paper provides a generalized framework and step by step analysis to retrieve useful information from memory.

Keywords—Cyber Forensics, Cyber Crime, RAM, Memory Forensics, Trojan Malware, Kali Linux, Volatile data, Volatility tools

1. Introduction

Nowadays, Government agencies and private companies are attempting to protect themselves from cyber-attacks with digital defense techniques like encryption, firewalls and signature scanning, etc. [1]. Memory forensics is defined as the process of analyzing volatile memory in a computer system. Information security professionals conduct memory forensics to investigate and identify or malicious behaviors that do not leave easily detectable tracks on hard drive data [2].

Memory forensic is helpful to analyze physical memory, RAM, to collect the evidence by recovering the data from the seized device that was used during the crime [3]. Memory forensic is also helpful to provide visibility into the runtime state of the system, and, the memory (RAM) must be analyzed for forensic information [4].

Memory forensics is about capturing the profile as well as the memory contents and can add an invaluable resource to incident response, malware analysis, and digital forensics capabilities. It is allowing an individual to determine what has already happened, what is presently happening, and what would happen with further infection through malware [5]. This paper focuses on use of memory forensic to recover the data from the system. Firstly, we need to acquire the memory image of the suspect machine and analyses it to get the crucial information about the system.

In this paper, we applied image sample infected with R2D2 malware is analyzed by using Memory Forensic tool 'Volatility'. The reason for choosing R2D2 Malware sample is based on the popularity and the financial impact on an environment. Memory sample can download in Github.com. This analyses demonstrate a practical approach which can be helpful in the detecting the advanced threats.

The objective of this paper is to trace the malware that is running silently and infecting the host. The system will provide evidence of the Trojan malware symptom and find out the indicators of the Compromise (IOC's) for the conformity of the malware detection.

The structure of the paper is as follows: Section 2 discusses the concept of memory forensics. Section 3 explains an overview of volatility tools and techniques. In Section 4, demonstrate how memory forensic can be useful in digging the traces of the Malware. Section 5 provides analysis and results of the system. Finally, conclusions and future work are presented in Section 6.

2. Memory Forensics

Memory forensics involves analyzing the data stored in the physical memory at operating system runtime. Its primary application is in the investigation of advanced computer attacks which are quiet enough to avoid leaving data on the computer hard drive. Consequently, the memory (RAM) must be analyzed for forensic information. Every function performed by an application or operating system results in a special kind of change to the random access memory. These changes often stay for a long time after completion of the operation, significantly storing them, memory forensics provides extraordinary visibility into the runtime state of the system, such as which processes were running, open network connections, and recently executed commands. Individuals can perform an extraction of these artifacts that is totally independent of the machine being investigated [6].

Memory forensics is forensic analysis of a computer's memory dump and its primary application is investigation of advanced computer attacks which are stealthy enough to avoid leaving data on the computer's hard drive. Consequently, the memory (RAM) must be analyzed for forensic information [7].

Figure (1) shows the process of memory forensics techniques. The first step is acquiring image from the victim's system. In second step, memory structure are

parsed and segments are identified for a process. Finally, outliers are searched like unlinked processes, hooks, known anomalies etc. Finally, outliers are searched like unlinked processes, hooks, known anomalies etc. It is also analyzing the data for evidence collection. Memory forensic is about capturing the memory contents which is a great tool for incident response and malware analysis.

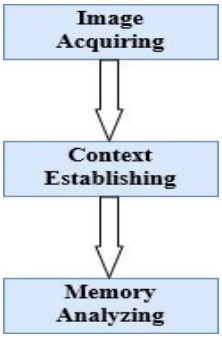

Figure 1. Process of Memory Forensics

3. Volatility Tools and Techniques

Volatility is a popular memory forensic tool. It is a single, cohesive framework that analyzes RAM dumps from Linux, 32- and 64-bit windows, Mac, and Android systems. The modular design of Volatility allows it to easily support new operating systems and architectures as they are released. So, all devices are targets. It doesn't limit the forensic capabilities to just Windows computers. Furthermore, it is an open source written in Python and has extensible and scriptable API with unparalleled feature sets and comprehensive coverage of file formats [8] [9].

Volatility is an open source memory forensics framework for incident response and malware analysis. It is written in Python and supports Microsoft Windows, Mac OS X and Linux. Volatility is analyzing RAM in 32 bit/64 bit systems. It can analyze raw dumps, crash dumps, VMware dumps (.vmem), and virtual box dumps which the data can be recovered [6]. In this paper, we applied R2D2.vmem malware attack image file which run in Kali Linux to analysis the system.

4. Demonstration

Sample image infected with R2D2 malware is chosen to demonstrate how memory forensic can be useful in digging the trace of the malware. To determine the characteristics of the memory dump, we used the image info plugin of volatility. The image info plugin identifies the Window operating system version, the service pack, and the architecture of the system.

The image info plugin also shows the date and time when the memory sample was collected as shown in Figure (2).

In figure 2, by using information of the suggested profiles that it suit with profile WinXPSP2x86 or WinXPSP3x86. Now, we can summarize the correct profile to identify the operating system.

The first step is collecting the processes those were running on the machine as follow in Figure 3. It will give us the information about all the running tasks in the memory.

The next step is to find out what application is running at that time the dump was taken in Figure 4. The **pstree plugin** displays the running processes in a parent-child structure. The process reader_sl.exe with PID 228 seems out of place because we have never heard of that process. However, it is too soon to draw conclusions.

Another step is to check which processes are trying to hide themselves by using **psview plugin** that shown in Figure 5. We can see that there are no hidden processes. If there were hidden processes, the corresponding attribute in the **pslist or psscan** columns would be False. Next step is to determine the network activity at the time when the memory dump was taken.

We can use the plugins **connscan** (Figure 6) and **sockets** (Figure 7). The **connscan** plugin scans the image for TCP connections and it also check whether any process is trying to connect the remoteIP's.

However, these connections are not guaranteed to be active. The **sockets** plugin prints a list of open sockets.

We can see that a process with PID 1956 is communicating with a remote address, 172.16.98.1:6666 on port 1026. Process ID 1956 is explorer.exe and reader_sl.exe is a child of explorer.exe in previous **pslist** and **pstree** plugin. From this point, reader_sl.exe is a definitely suspect.

Another step is to find which commands were run in system as shown in Figure 8. Two commands which **sc query malwar** and **sc query malware** were run on the console as shown on Figure 8.The first command sc query malwar may have just been a typo mistake. It seems like the attacker / user was searching for a service named malwar or malware. The **cmdline** plugin provides process command-line arguments. It is the complete command which was used to launch each process.

We can see that reader_sl.exe that we suspect as the malware is actually installed in "C:\Program Files\Adobe\Reader 9.0\Reader\Reader_sl.exe" and triggered without any parameter. We can compare the application that is triggered with parameter such as Command line: C:\WINDOWS\system32\svchost.exe -k LocalService is shown on Figure 9.

Finally, we need to analyze the binary of reader_sl.exe. We can extract that specific binary from the memory dump.

Volatility extract the binary from the memory and name the binary with executable.exe on above Figure 10. By using executable file, we can analyze what kind of malware it is.

5. Analysis and Results

Now, we have collected the indicator of commands to point the malware infection. We will check the

executable.228.exe file at virustotal.com as follow figure 11. Now we can conclude that the binary is malware and **four engine** from virus total detected this is malicious. We should know what this malware is trying to do or how the malware could install itself in the host.

Next analysis is to do dump the memory area (228.dmp) that the malware use. We can narrow down our search for this particular malware only in figure 12.

There are lots of references to strings containing malware. With this information, we can say with a fair degree of confidence that reader_sl.exe is look like malicious. From this analysis, we can conclude that the sample image may contain sensitive information related to your organization such as usernames and passwords.

```
root@kali:~/Desktop# vol.py -f 0zapftis.vmem imageinfo
Volatility Foundation Volatility Framework 2.6.1
INFO    : volatility.debug    : Determining profile based on KDBG search...
          Suggested Profile(s) : WinXPSP2x86, WinXPSP3x86 (Instantiated with WinXPSP2x86)
                     AS Layer1 : IA32PagedMemoryPae (Kernel AS)
                     AS Layer2 : FileAddressSpace (/root/Desktop/0zapftis.vmem)
                      PAE type : PAE
                           DTB : 0x319000L
                          KDBG : 0x80544ce0L
          Number of Processors : 1
     Image Type (Service Pack) : 2
                KPCR for CPU 0 : 0xffdff000L
             KUSER_SHARED_DATA : 0xffdf0000L
           Image date and time : 2011-10-10 17:06:54 UTC+0000
     Image local date and time : 2011-10-10 13:06:54 -0400
```

Figure 2. Capture Window Operating System

```
root@kali:~/Desktop# vol.py WinXPSP3x86 -f  0zapftis.vmem pslist
Volatility Foundation Volatility Framework 2.6.1
Offset(V)  Name                    PID   PPID   Thds     Hnds   Sess  Wow64 Start                          Exit
---------- -------------------- ------ ------ ------ -------- ------ ------ ------------------------------ ----
0x819cc830 System                    4      0     55      162 ------      0
0x81945020 smss.exe                536      4      3       21 ------      0 2011-10-10 17:03:56 UTC+0000
0x816c6020 csrss.exe               608    536     11      355      0      0 2011-10-10 17:03:58 UTC+0000
0x813a9020 winlogon.exe            632    536     24      533      0      0 2011-10-10 17:03:58 UTC+0000
0x816da020 services.exe            676    632     16      261      0      0 2011-10-10 17:03:58 UTC+0000
0x813c4020 lsass.exe               688    632     23      336      0      0 2011-10-10 17:03:58 UTC+0000
0x81772ca8 vmacthlp.exe            832    676      1       24      0      0 2011-10-10 17:03:59 UTC+0000
0x8167e9d0 svchost.exe             848    676     20      194      0      0 2011-10-10 17:03:59 UTC+0000
0x817757f0 svchost.exe             916    676      9      217      0      0 2011-10-10 17:03:59 UTC+0000
0x816c6da0 svchost.exe             964    676     63     1058      0      0 2011-10-10 17:03:59 UTC+0000
0x815daca8 svchost.exe            1020    676      5       58      0      0 2011-10-10 17:03:59 UTC+0000
0x813aeda0 svchost.exe            1148    676     12      187      0      0 2011-10-10 17:04:00 UTC+0000
0x817937e0 spoolsv.exe            1260    676     13      140      0      0 2011-10-10 17:04:00 UTC+0000
0x81754990 VMwareService.e        1444    676      3      145      0      0 2011-10-10 17:04:00 UTC+0000
0x8136c5a0 alg.exe                1616    676      7       99      0      0 2011-10-10 17:04:01 UTC+0000
0x815c4da0 wscntfy.exe            1920    964      1       27      0      0 2011-10-10 17:04:39 UTC+0000
0x813bcda0 explorer.exe           1956   1884     18      322      0      0 2011-10-10 17:04:39 UTC+0000
0x816d63d0 VMwareTray.exe          184   1956      1       28      0      0 2011-10-10 17:04:41 UTC+0000
0x8180b478 VMwareUser.exe          192   1956      6       83      0      0 2011-10-10 17:04:41 UTC+0000
0x818233c8 reader_sl.exe           228   1956      2       26      0      0 2011-10-10 17:04:41 UTC+0000
0x815e7be0 wuauclt.exe             400    964      8      173      0      0 2011-10-10 17:04:46 UTC+0000
0x817a34b0 cmd.exe                 544   1956      1       30      0      0 2011-10-10 17:06:42 UTC+0000
```

Figure 3. Output of the Process List

Consoles command print the output all of the commands. We can see that there is a running service named *malware* which is of type, *KERNEL_DRIVER* at Figure 13. It means that the service is running in kernel mode and a malicious driver in the kernel does not bode well.

In this paper, firstly we extracted the malware behavior and realized the malware is existing in this image sample. To analyze the malware sample, we used volatility tools in Kali Linux. This paper mainly focus on finding malware at image sample by using analyzing tool. It is not include malware prevention and deletion techniques. Later, we will analyze the other malware sample to compare with this system.

```
root@kali:~/Desktop# vol.py WinXPSP3x86 -f  0zapftis.vmem pstree
Volatility Foundation Volatility Framework 2.6.1
Name                                                  Pid   PPid   Thds   Hnds Time
-------------------------------------------------- ------ ------ ------ ------ ----
 0x819cc830:System                                      4      0     55    162 1970-01-01 00:00:00 UTC+0000
. 0x81945020:smss.exe                                 536      4      3     21 2011-10-10 17:03:56 UTC+0000
.. 0x816c6020:csrss.exe                               608    536     11    355 2011-10-10 17:03:58 UTC+0000
.. 0x813a9020:winlogon.exe                            632    536     24    533 2011-10-10 17:03:58 UTC+0000
... 0x816da020:services.exe                           676    632     16    261 2011-10-10 17:03:58 UTC+0000
.... 0x817757f0:svchost.exe                           916    676      9    217 2011-10-10 17:03:59 UTC+0000
.... 0x81772ca8:vmacthlp.exe                          832    676      1     24 2011-10-10 17:03:59 UTC+0000
.... 0x816c6da0:svchost.exe                           964    676     63   1058 2011-10-10 17:03:59 UTC+0000
..... 0x815c4da0:wscntfy.exe                         1920    964      1     27 2011-10-10 17:04:39 UTC+0000
..... 0x815e7be0:wuauclt.exe                          400    964      8    173 2011-10-10 17:04:46 UTC+0000
.... 0x8167e9d0:svchost.exe                           848    676     20    194 2011-10-10 17:03:59 UTC+0000
.... 0x81754990:VMwareService.e                      1444    676      3    145 2011-10-10 17:04:00 UTC+0000
.... 0x8136c5a0:alg.exe                              1616    676      7     99 2011-10-10 17:04:01 UTC+0000
.... 0x813aeda0:svchost.exe                          1148    676     12    187 2011-10-10 17:04:00 UTC+0000
.... 0x817937e0:spoolsv.exe                          1260    676     13    140 2011-10-10 17:04:00 UTC+0000
.... 0x815daca8:svchost.exe                          1020    676      5     58 2011-10-10 17:03:59 UTC+0000
... 0x813c4020:lsass.exe                              688    632     23    336 2011-10-10 17:03:58 UTC+0000
 0x813bcda0:explorer.exe                             1956   1884     18    322 2011-10-10 17:04:39 UTC+0000
. 0x8180b478:VMwareUser.exe                           192   1956      6     83 2011-10-10 17:04:41 UTC+0000
. 0x817a34b0:cmd.exe                                  544   1956      1     30 2011-10-10 17:06:42 UTC+0000
. 0x816d63d0:VMwareTray.exe                           184   1956      1     28 2011-10-10 17:04:41 UTC+0000
. 0x818233c8:reader_sl.exe                            228   1956      2     26 2011-10-10 17:04:41 UTC+0000
```

Figure 4. Output of the Process List (Parent-Child)

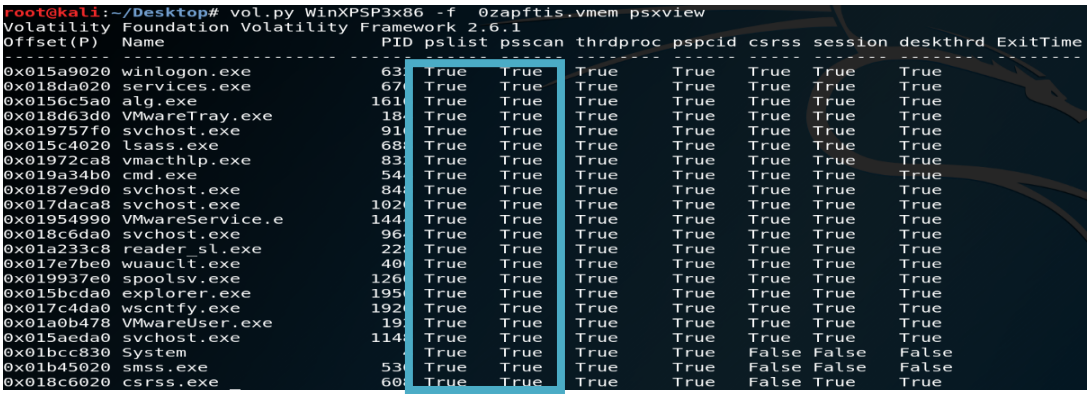

Figure 5. Processes List (no hidden)

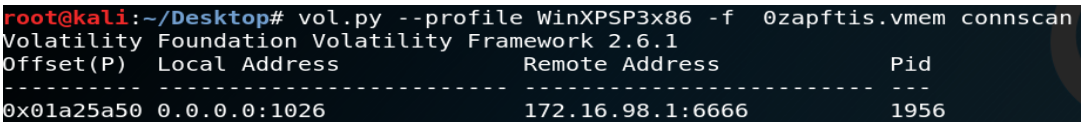

Figure 6. Image scan

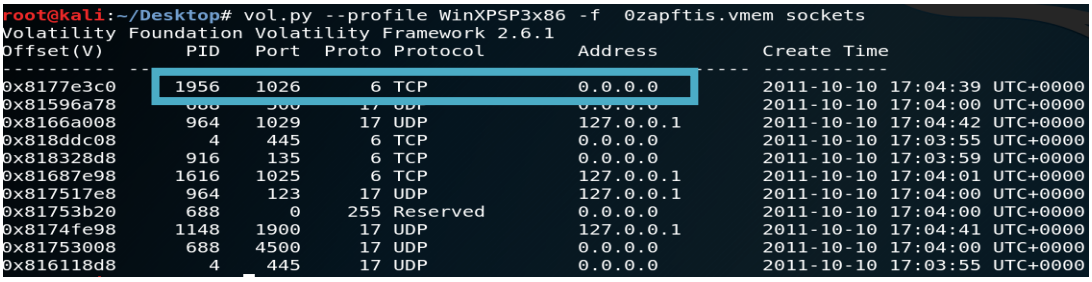

Figure 7. Open Sockets

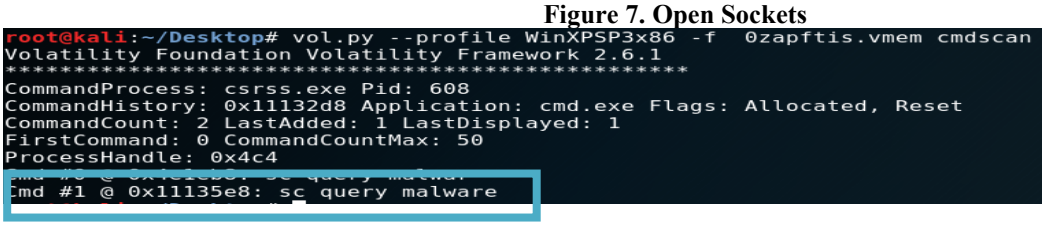

Figure 8. Show Commands

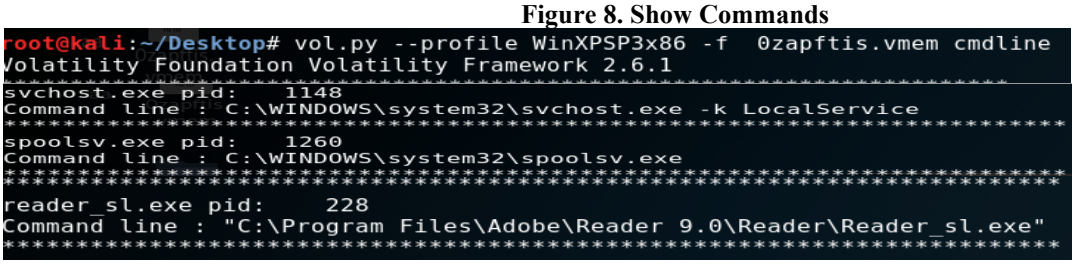

Figure 9. Command line Comparison

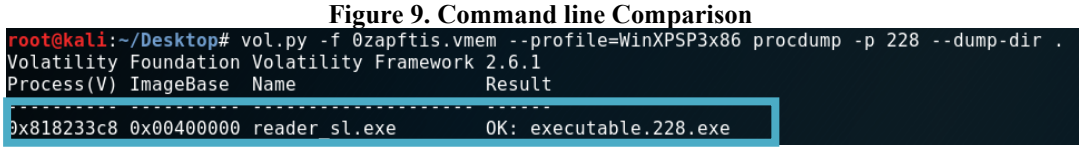

Figure 10. Extracting Memory Dump

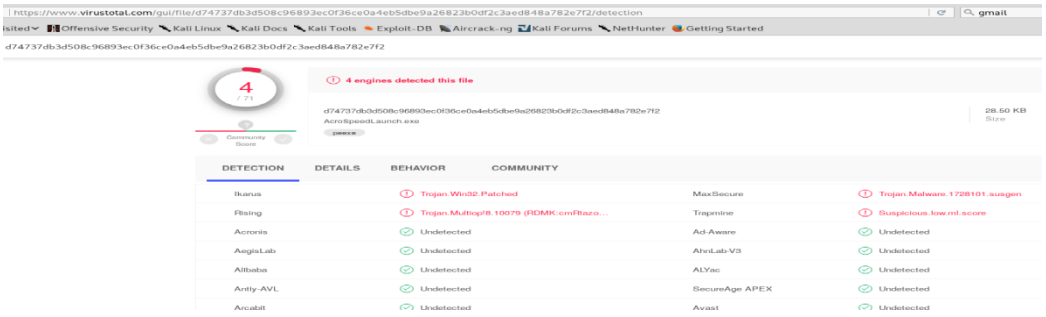

Figure 11. Detection of Malware

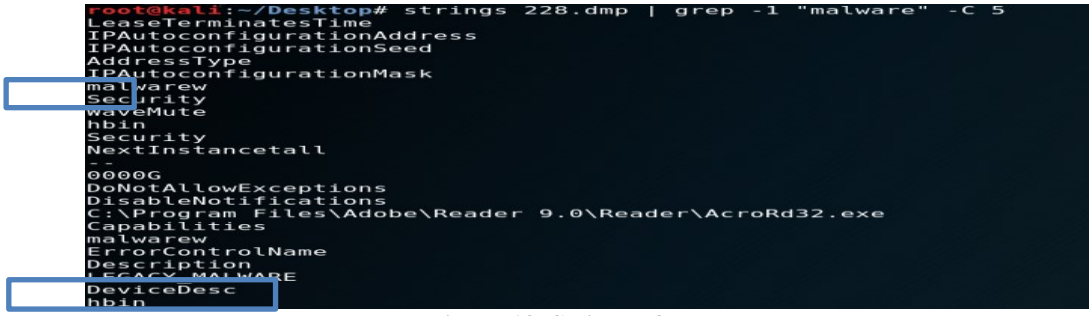

Figure 12. Strings of Memory Dump

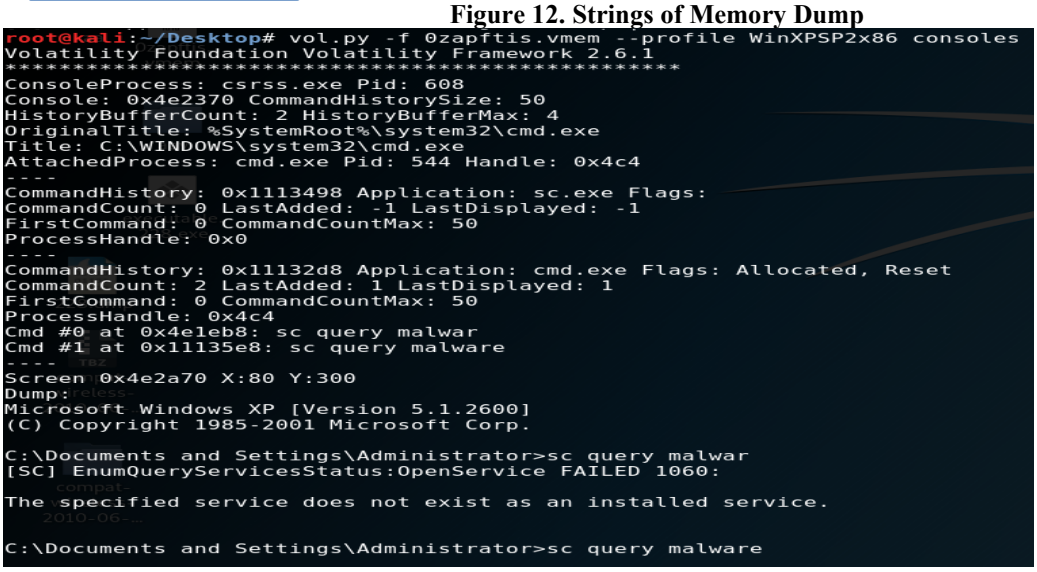

Figure 13. Output Commands

6. Conclusion and Future Work

This paper has demonstrated with memory forensics approach. Memory forensics is useful in conducting the stealthy volatile attacks which many time reside only in memory. We have presented a brief approach which we can proceed with the analysis for finding out traces of advanced volatile threats existing in the memory later. We can use the command line tool 'volatility' as well as knowledge and methodology of static and dynamic malware analysis. Some work can be done to help cyber investigation in detecting and analyzing malware from the RAM dump of the machine.

References

[1] Baliga, A., Ganapathy, V. and Iftode, L., "Detecting kernel-level root kits using data structure invariants", *IEEE Transactions on Dependable and Secure Computing*, 8(5), pp.670-684, 2011.

[2] Dr. Hardik Gohel, Dr.Himanshu Upadhyay, "Design of Advanced Cyber Threat Analysis Framework for Memory Forensics", *International Journal of Innovative Research in Computer and Communication Engineering*, 5, 2, p132-137, 2017.

3] Hardik Gohel. "Introduction to Network & Cyber Security", 2015.

[4] Lime Forensics, https://code.google.com /p/ lime- forensics/.

[5] Mital Parekh, Snehal Jani, "Memory forensic: Acquistion and Analysis of memory and its tools

comparison", *International Journal of Engineering Technologies and Maanagement Research*, Vol 5, Feb, p90-95, 2018.
[6] Pooja Salave, Atisha Wakdikar, "Memory Forensics: Tools Comparison", *International Journal of Science and Research (IJSR)*, 6, 6, p5-8, 2017.
[7] Reith M, Carr C, Gunsch G, "An examination of Digital Forensics Models", *International Journal of Digital Evidence*, 1, 3, p1–12, 2007.
[8] Vitou Pen, *Memory Forensic analysis (VOLAT ILITY)*.ppt
[9] Volatility framework, https://github .com/volat ility foundation/volatiliy/wiki.