COMP 8220 Computer and Memory Forensics

Tutorial 2 Memory Forensics

Volatile Memory Analysis

The volatile memory (RAM) represents an important source of volatile data that can help further the forensics investigation. Whenever possible, volatile memory should be imaged when responding initially to an incident. The memory image or dump can be analyzed later using adequate tools in order to extract volatile system and network information, such as running processes, open network connections, etc.

In the tutorial, we will practice volatility a memory dump captured from a machine infected by the **Zeus botnet.** You can download the dump posted here: https://drive.google.com/file/d/0B1xnRxT-Y8DMUENnSGRwSkN0TlU/view?usp=sharing

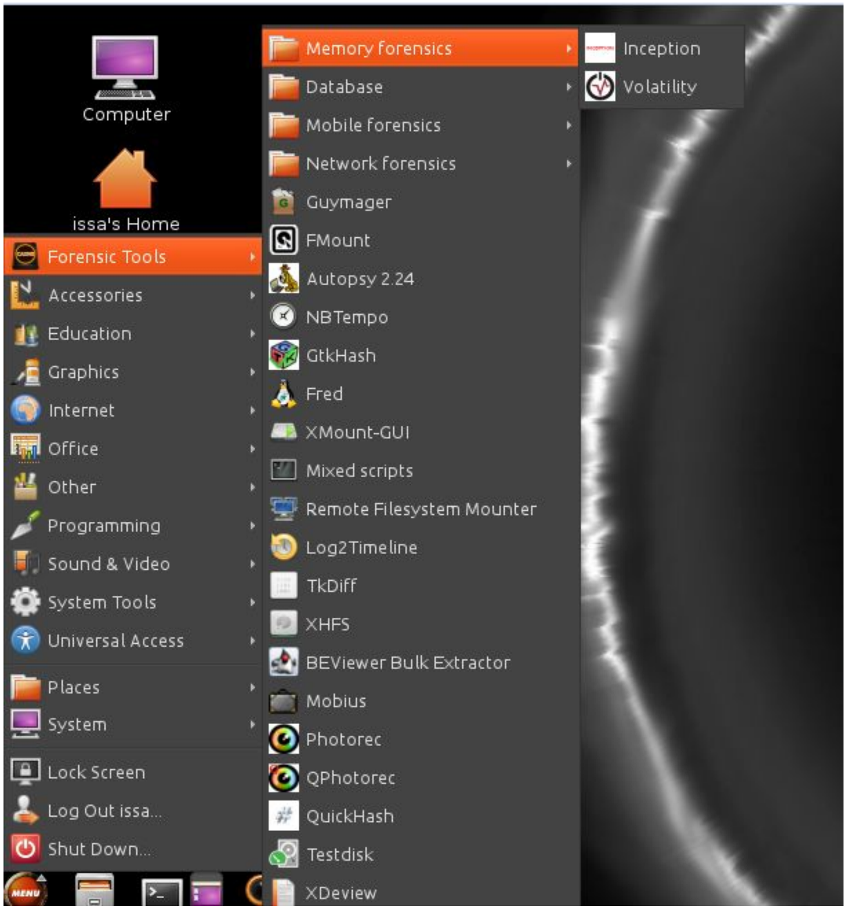
The file is a zip file; after extracting it, you'll get a file named **zeus.vmem**.

We will use in this tutorial, a memory analysis tool named **Volatility** that is available in Caine and Kali. Volatility is a well-known collection of open source tools used to extract digital artifacts from volatile memory. The Volatility Framework currently provides among others the following extraction capabilities for memory samples:

- Image information (date, time, CPU count)
- Running processes, Process SIDs and environment variables
- Open network sockets and Open network connections
- Command histories (cmd.exe) and console input/output buffersUsing Volatility Framework

To start Volatility in Caine, select in the Menu:

Forensics Tools>Memory Forensics>Volatility



This will bring up a command line window.

Volatility runs on python. The most basic Volatility commands are constructed as follows:

python vol.py [plugin] -f [image] --profile=[profile]

Where *plug-in* with is the name of an external plug-in (if any) to use, the *image* is the file path to your memory image, and *profile* is the name of the memory profile.

The first step in running volatility against a memory sample is to identify the memory profile. This can be done by typing the following command (assuming that my memory sample is located under **/home/issa/Caine_Evidence/zeus**):

python vol.py -f /home/issa/Caine_Evidence/zeus/zeus.vmem imageinfo

```
issa@caine-vm:/usr/share/caine/pacchetti/volatility$ python vol.py -f /home/issa
/Caine_Evidence/zeus/zeus.vmem imageinfo
Volatility Foundation Volatility Framework 2.4
Determining profile based on KDBG search...

          Suggested Profile(s) : WinXPSP2x86, WinXPSP3x86 (Instantiated with Win
XPSP2x86)
                     AS Layer1 : IA32PagedMemoryPae (Kernel AS)
                     AS Layer2 : FileAddressSpace (/home/issa/Caine_Evidence/zeu
s/zeus.vmem)
                      PAE type : PAE
                           DTB : 0x319000L
                          KDBG : 0x80544ce0
          Number of Processors : 1
     Image Type (Service Pack) : 2
                KPCR for CPU 0 : 0xffdff000
             KUSER_SHARED_DATA : 0xffdf0000
           Image date and time : 2010-08-15 19:17:56 UTC+0000
     Image local date and time : 2010-08-15 15:17:56 -0400
issa@caine-vm:/usr/share/caine/pacchetti/volatility$
```

Among other things, the *imageinfo* output tells you the suggested profile that you should pass as the parameter to --profile=PROFILE; there may be more than one profile suggestion if profiles are closely related. You can figure out which one is more appropriate by checking the "Image Type" field, which is blank for Service Pack 0 and filled in for other Service Packs.
In our case, the Image Type is indicating service pack 2, which points to WinXPSP2x86 as profile.
Using such information, we can execute different volatility commands to gather as much information as possible. I suggest exploring the man page to find out available commands and the possibility of the tool.
A cheat sheet summarizing useful commands for malware analysis can be downloaded here:
https://blogs.sans.org/computer-forensics/files/2012/04/Memory-Forensics-Cheat-Sheet-v1_2.pdf
All the remaining commands will require specifying the location of the memory file and the profile, which can be very cumbersome. To avoid that, Volatility provides environments that can be set initially (after finding out the profile):

export VOLATILITY_LOCATION=file:///home/issa/Caine_Evidence/zeus/zeus.vmem
export VOLATILITY_PROFILE=WinXPSP2x86

```
issa@caine-vm:/usr/share/caine/pacchetti/volatility$ export VOLATILITY_PROFILE=W
inXPSP2x86
issa@caine-vm:/usr/share/caine/pacchetti/volatility$ export VOLATILITY_LOCATION=
file:///home/issa/Caine_Evidence/zeus/zeus.vmem
```

After setting the environment variables, you can use the commands without specifying the filename or profile; this will be valid as long the command line window is open.

Identifying Rogue Processes

One of the most useful commands in acquiring volatile information is listing running processes.
You can list the processes of a system by using the **pslist** plugin as follows:

python vol.py pslist

```
issa@caine-vm:/usr/share/caine/pacchetti/volatility$ python vol.py pslist
Volatility Foundation Volatility Framework 2.4
Offset(V)  Name                    PID   PPID   Thds     Hnds   Sess  Wow64 Star
t                          Exit
---------- -------------------- ------ ------ ------ -------- ------ ------ ----
-------------------------- ------------------------------
0x810b1660 System                    4      0     58      379 ------      0

0xff2ab020 smss.exe                544      4      3       21 ------      0 2010
-08-11 06:06:21 UTC+0000
0xff1ecda0 csrss.exe               608    544     10      410      0      0 2010
-08-11 06:06:23 UTC+0000
0xff1ec978 winlogon.exe            632    544     24      536      0      0 2010
-08-11 06:06:23 UTC+0000
0xff247020 services.exe            676    632     16      288      0      0 2010
-08-11 06:06:24 UTC+0000
0xff255020 lsass.exe               688    632     21      405      0      0 2010
-08-11 06:06:24 UTC+0000
0xff218230 vmacthlp.exe            844    676      1       37      0      0 2010
-08-11 06:06:24 UTC+0000
```

This displays a doubly-linked list of processes. It does not detect hidden or unlinked processes.

Also, if you see processes with 0 threads and 0 handles, the process may not actually still be active. The columns display the offset, process name, process ID, the parent process ID, number of threads, number of handles, and date/time when the process started. The offset is a virtual address by default, but the physical offset can be obtained with the -P switch:

python vol.py pslist -P

By analyzing the process list displayed above, nothing unusual appears; at least in the first reading. All the processes sound normal as they the established baseline. Further analysis must be carried out.

To view the process listing in tree form, use the **pstree** plugin as follows

python vol.py pstree

```
issa@caine-vm:/usr/share/caine/pacchetti/volatility$ python vol.py pstree
Volatility Foundation Volatility Framework 2.4
Name                                                  Pid   PPid   Thds   Hnds Time
-------------------------------------------------- ------ ------ ------ ------ ----
 0x810b1660:System                                      4      0     58    379 1970-01-01 00:00:00 UTC+0000
. 0xff2ab020:smss.exe                                 544      4      3     21 2010-08-11 06:06:21 UTC+0000
.. 0xff1ec978:winlogon.exe                            632    544     24    536 2010-08-11 06:06:23 UTC+0000
... 0xff255020:lsass.exe                              688    632     21    405 2010-08-11 06:06:24 UTC+0000
... 0xff247020:services.exe                           676    632     16    288 2010-08-11 06:06:24 UTC+0000
.... 0xff1b8b28:vmtoolsd.exe                         1668    676      5    225 2010-08-11 06:06:35 UTC+0000
..... 0xff224020:cmd.exe                              124   1668      0 ------ 2010-08-15 19:17:55 UTC+0000
.... 0x80ff88d8:svchost.exe                           856    676     29    336 2010-08-11 06:06:24 UTC+0000
.... 0xff1d7da0:spoolsv.exe                          1432    676     14    145 2010-08-11 06:06:26 UTC+0000
.... 0x80fbf910:svchost.exe                          1028    676     88   1424 2010-08-11 06:06:24 UTC+0000
..... 0x80f60da0:wuauclt.exe                         1732   1028      7    189 2010-08-11 06:07:44 UTC+0000
..... 0x80f94588:wuauclt.exe                          468   1028      4    142 2010-08-11 06:09:37 UTC+0000
..... 0xff364310:wscntfy.exe                          888   1028      1     40 2010-08-11 06:06:49 UTC+0000
.... 0xff217560:svchost.exe                           936    676     11    288 2010-08-11 06:06:24 UTC+0000
.... 0xff143b28:TPAutoConnSvc.e                      1968    676      5    106 2010-08-11 06:06:39 UTC+0000
..... 0xff38b5f8:TPAutoConnect.e                     1084   1968      1     68 2010-08-11 06:06:52 UTC+0000
.... 0xff22d558:svchost.exe                          1088    676      7     93 2010-08-11 06:06:25 UTC+0000
```

This enumerates processes using the same technique as **pslist**, so it will also not show hidden or unlinked processes.

Reviewing Network Artifacts

Another important piece of volatile information that can be obtained using volatility is active network connections, which can be viewed using the **connections** plugin as follows:

python vol.py connections

```
issa@caine-vm:/usr/share/caine/pacchetti/volatility$ python vol.py connections
Volatility Foundation Volatility Framework 2.4
Offset(V)  Local Address             Remote Address            Pid
---------- ------------------------- ------------------------- ---
issa@caine-vm:/usr/share/caine/pacchetti/volatility$
```

This will display a singly-linked list of connection structures including the offset, local address, remote address, and PID. Note that the above list being empty indicates possibly there were no networking activities at the time of imaging. The offset obtained using the above is a virtual offset (like in the case of running processes). The physical offset is obtained with the -P switch as follows:

python vol.py -P

The fact that there were no active network connections at the time of capture, doesn't mean the no network connections were established at all.

One of the key characteristics of malware such as botnets is to connect to external locations, especially for command and control or to download some malware code, etc. We can list recent network connections using the **connscan** plugin as follows:

python vol.py connscan

```
issa@caine-vm:/usr/share/caine/pacchetti/volatility$ python vol.py connscan
Volatility Foundation Volatility Framework 2.4
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
issa@caine-vm:/usr/share/caine/pacchetti/volatility$
```

This shows 2 connections established by the machine to the remote IP address 193.104.41.75

Both connections were made by a process with PID=856. Since this process is connecting to port 80, one would expect that it belongs to an Internet browser (chrome, etc.).

By checking the process list (returned by pslist), it corresponds to the process **svchost.exe**, which is a service.

```
issa@caine-vm:/usr/share/caine/pacchetti/volatility$ python vol.py pslist
Volatility Foundation Volatility Framework 2.4
Offset(V)  Name                    PID   PPID   Thds     Hnds   Sess  Wow64 Start                          Exit

---------- -------------------- ------ ------ ------ -------- ------ ------ ------------------------------ ----------
--
0x810b1660 System                    4      0     58      379 ------      0

0xff2ab020 smss.exe                544      4      3       21 ------      0 2010-08-11 06:06:21 UTC+0000

0xff1ecda0 csrss.exe               608    544     10      410      0      0 2010-08-11 06:06:23 UTC+0000

0xff1ec978 winlogon.exe            632    544     24      536      0      0 2010-08-11 06:06:23 UTC+0000

0xff247020 services.exe            676    632     16      288      0      0 2010-08-11 06:06:24 UTC+0000

0xff255020 lsass.exe               688    632     21      405      0      0 2010-08-11 06:06:24 UTC+0000

0xff218230 vmacthlp.exe            844    676      1       37      0      0 2010-08-11 06:06:24 UTC+0000

0x80ff88d8 svchost.exe             856    676     29      336      0      0 2010-08-11 06:06:24 UTC+0000

0xff217560 svchost.exe             936    676     11      288      0      0 2010-08-11 06:06:24 UTC+0000
```

This is confirmed by its parent process (PID=676) which corresponds to **service.exe**. Now, we have a service connecting to port 80; which is abnormal. This means that there is a great chance that the process 856 is malicious.

We can check open sockets on the machine using the **sockscan** plugin as follows:

python vol.py sockscan

```
issa@caine-vm:/usr/share/caine/pacchetti/volatility$ python vol.py sockscan
Volatility Foundation Volatility Framework 2.4
Offset(P)       PID   Port  Proto Protocol        Address         Create Time
---------- -------- ------ ------ --------------- --------------- -----------
0x007c0a20     1148   1900     17 UDP             172.16.176.143  2010-08-15 19:15:43 UTC+0000
0x01120c40        4    445     17 UDP             0.0.0.0         2010-08-11 06:06:17 UTC+0000
0x01131930     1088   1025     17 UDP             0.0.0.0         2010-08-11 06:06:38 UTC+0000
0x01134008        4      0     47 GRE             0.0.0.0         2010-08-11 06:08:00 UTC+0000
0x011568a8        4    138     17 UDP             172.16.176.143  2010-08-15 19:15:43 UTC+0000
0x0115f128      936    135      6 TCP             0.0.0.0         2010-08-11 06:06:24 UTC+0000
0x02daad28      216   1026      6 TCP             127.0.0.1       2010-08-11 06:06:39 UTC+0000
0x04863458        4    139      6 TCP             172.16.176.143  2010-08-15 19:15:43 UTC+0000
0x04864578     1028     68     17 UDP             172.16.176.143  2010-08-15 19:17:26 UTC+0000
0x04864a08        4    137     17 UDP             172.16.176.143  2010-08-15 19:15:43 UTC+0000
0x04a4be98        4   1033      6 TCP             0.0.0.0         2010-08-11 06:08:00 UTC+0000
0x04a51d28     1028   1058      6 TCP             0.0.0.0         2010-08-15 19:17:56 UTC+0000
0x04be7008        4    445      6 TCP             0.0.0.0         2010-08-11 06:06:17 UTC+0000
0x05dee200     1028    123     17 UDP             127.0.0.1       2010-08-15 19:15:43 UTC+0000
0x05e33d68     1148   1900     17 UDP             127.0.0.1       2010-08-15 19:15:43 UTC+0000
0x05f44008      688    500     17 UDP             0.0.0.0         2010-08-11 06:06:35 UTC+0000
0x05f48008     1028    123     17 UDP             127.0.0.1       2010-08-15 19:17:56 UTC+0000
0x06236e98     1028     68     17 UDP             172.16.176.143  2010-08-15 19:17:56 UTC+0000
0x06237b70      688      0    255 Reserved        0.0.0.0         2010-08-11 06:06:35 UTC+0000
0x06450478      856  29220      6 TCP             0.0.0.0         2010-08-15 19:17:27 UTC+0000
```

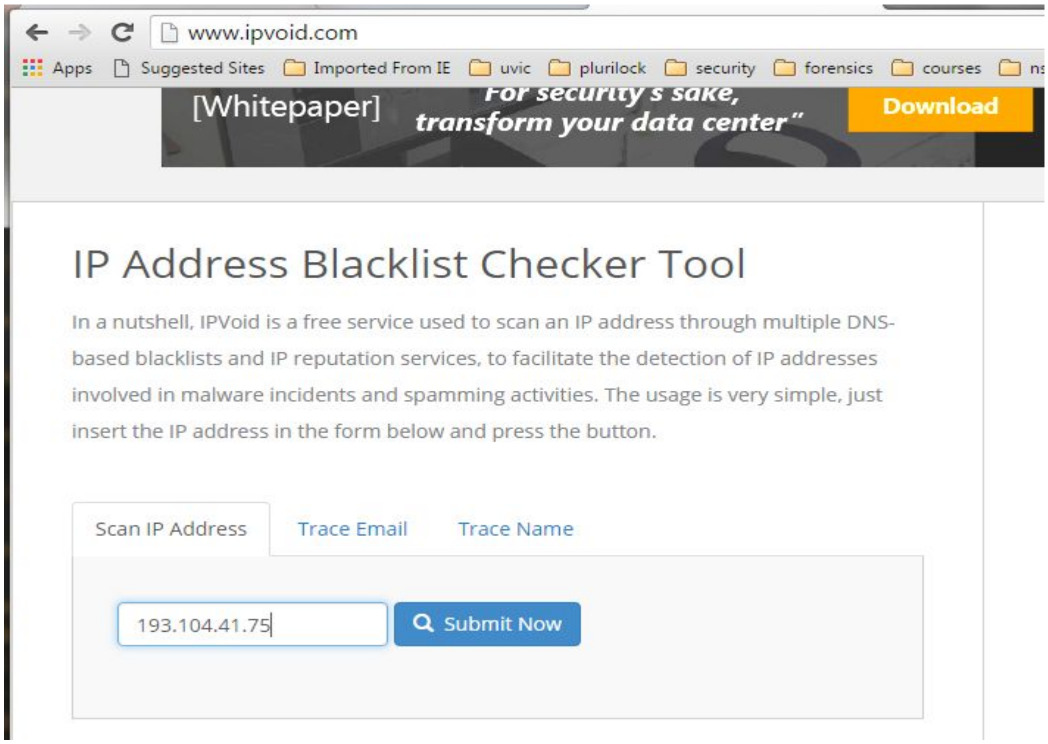
By analyzing the sockets, it may be possible to build the incident timeline, especially by focusing on entries relevant to the suspicious processes.

We can find out more about the remote site to which the suspicious connected. Using an online IP geo-location service, e.g., https://www.iplocation.net/, we obtain the following:

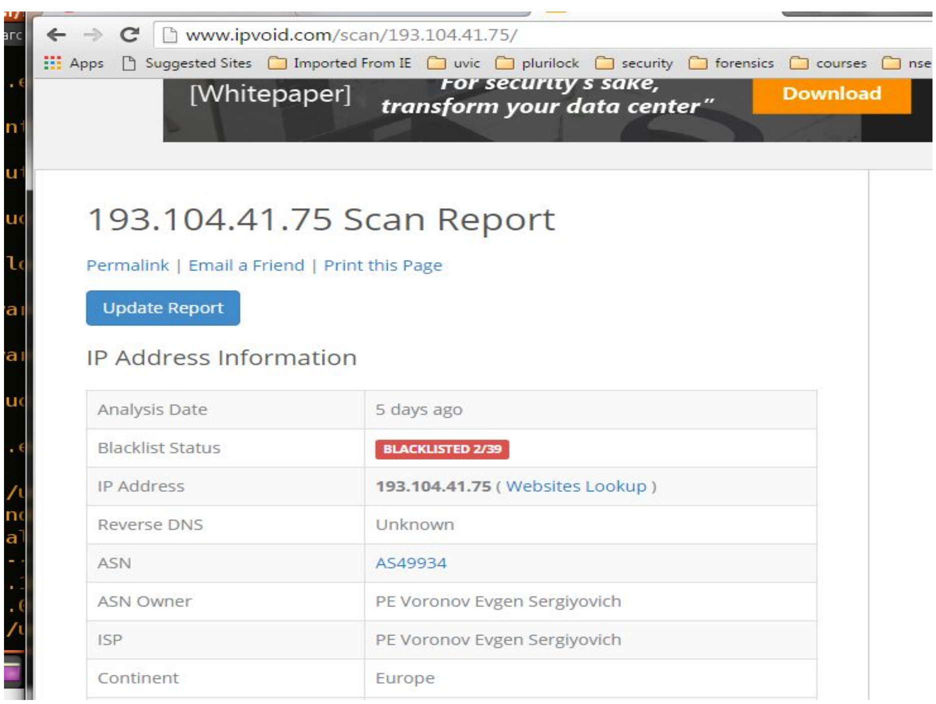
Geolocation data from DB-IP (Product: Full, 2016-5-2)

IP Address	Country	Region	City
193.104.41.75	Moldova	Stînga Nistrului	Tiraspol
ISP	**Organization**	**Latitude**	**Longitude**
PE Voronov Evgen Sergiyovich	PE Voronov Evgen Sergiyovich	46.8363	29.6144

The IP address is located in Moldova. We can check further if it is black listed, which may be a strong indicator that it is a known malicious site.



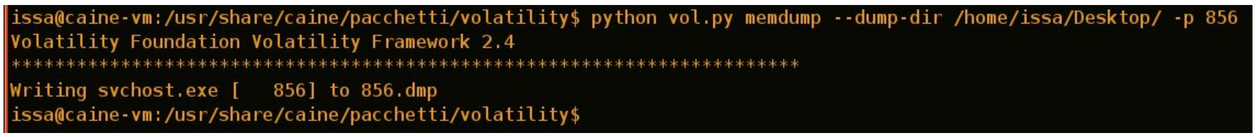
Searching the IP at www.ipvoid.com, corroborates our suspicion.



Dumping Suspicious Processes and Drivers

We can extract and further analyze the memories for suspicious processes. A memory dump for process with PID=856 can be extracted using the **memdump** plugin as follows:

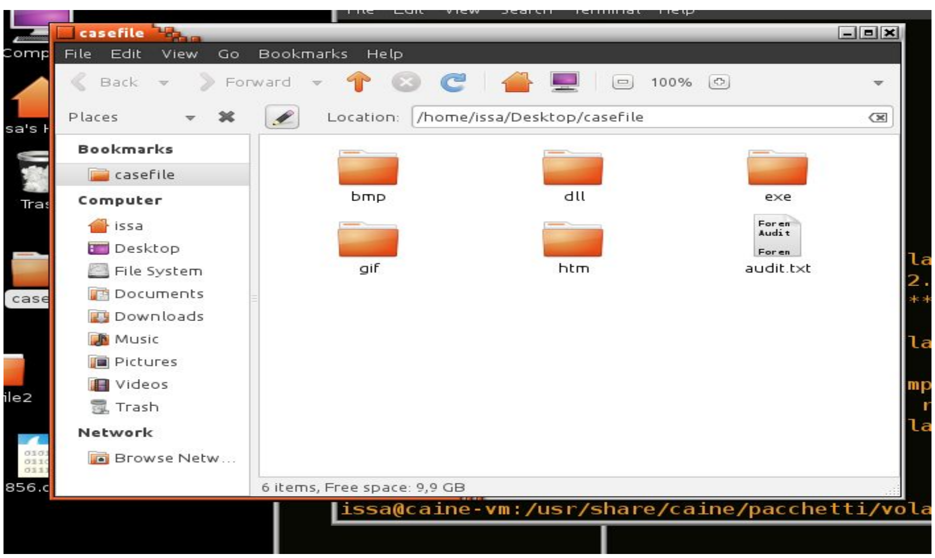
python vol.py memdump --dump-dir=/home/issa/Desktop/ -p 856



The memory will be generated as a file 856.dmp and stored at the specified location. Next, we can extract files from the memory dump (if there are any), using a file carving tool such as **foremost**, which is a command line tool available in Kali and Caine.

foremost -i /home/issa/Desktop/856.dmp -o /home/issa/Desktop/casefile/

The extracted files will be stored at the specified location.



The Foremost report, audit.txt, is located in the output directory.

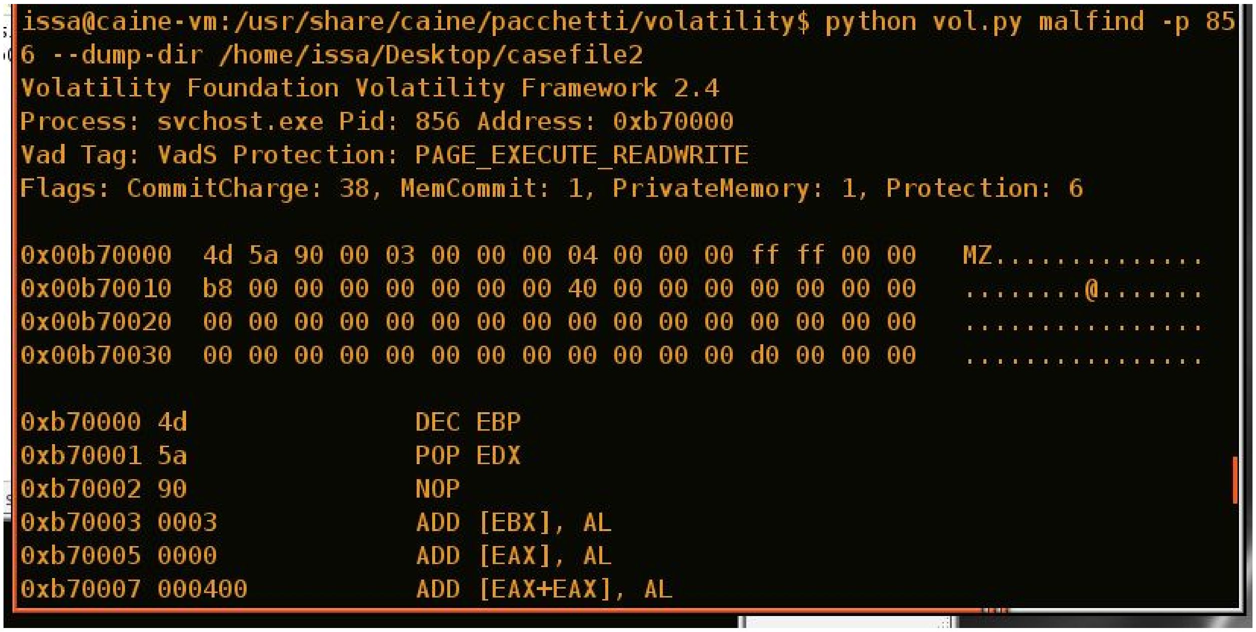
```
Foremost version 1.5.7 by Jesse Kornblum, Kris Kendall, and Nick Mikus
Audit File

Foremost started at Thu May 12 04:43:50 2016
Invocation: foremost -i /home/issa/Desktop/856.dmp -o /home/issa/Desktop/casefile/
Output directory: /home/issa/Desktop/casefile
Configuration file: /usr/local/etc/foremost.conf
----------------------------------------------------------------
File: /home/issa/Desktop/856.dmp
Start: Thu May 12 04:43:50 2016
Length: 60 MB (63823872 bytes)

Num  Name (bs=512)         Size  File Offset     Comment

0:  00068090.gif           64 B       34862368      (10 x 10)
1:  00059408.bmp          27 KB       30417377      (128 x 256)
2:  00059630.bmp          21 KB       30530770      (7085304 x 1)
3:  00062873.bmp          27 KB       32191105      (128 x 256)
4:  00018136.htm          218 B        9285845
5:  00018144.htm          218 B        9289941
6:  00018160.htm          218 B        9298133
7:  00108616.htm          218 B       55611605
8:  00108624.htm          218 B       55615701
9:  00108640.htm          218 B       55623893
10: 00001472.dll         202 KB         753664     08/04/2004 07:56:40
11: 00004208.dll         773 KB        2154496     08/04/2004 07:56:36
12: 00005824.exe          14 KB        2981888     08/04/2004 06:14:46
13: 00005976.dll           2 MB        3059712     08/04/2004 07:56:41
```

```
74: 00118168.exe           9 KB       60502016      08/17/2001 20:57:58
75: 00118200.exe          11 KB       60518400      11/13/2008 02:50:11
76: 00118800.exe          15 KB       60825600      08/04/2004 05:59:06
77: 00118848.exe           9 KB       60850176      08/17/2001 20:55:29
78: 00119008.exe          15 KB       60932096      08/04/2004 06:07:47
79: 00119184.exe          10 KB       61022208      08/17/2001 20:53:19
80: 00119296.dll           6 KB       61079552      08/17/2001 20:49:10
81: 00119312.exe           4 KB       61087744      08/17/2001 21:07:23
82: 00119336.exe           4 KB       61100032      09/29/2008 06:49:38
83: 00119352.dll           4 KB       61108224      08/17/2001 21:02:58
84: 00119368.exe           7 KB       61116416      08/17/2001 20:49:37
85: 00119384.exe           4 KB       61124608      08/17/2001 20:57:28
86: 00119416.exe           7 KB       61140992      02/09/2010 23:19:29
87: 00119616.exe           3 KB       61243392      08/17/2001 20:53:12
88: 00119656.exe           3 KB       61263872      08/17/2001 20:59:40
89: 00120056.exe         328 KB       61468672      08/04/2004 06:14:44
Finish: Thu May 12 04:43:51 2016

90 FILES EXTRACTED

gif:= 1
bmp:= 3
htm:= 6
exe:= 80
----------------------------------------------------------------

Foremost finished at Thu May 12 04:43:51 2016
```

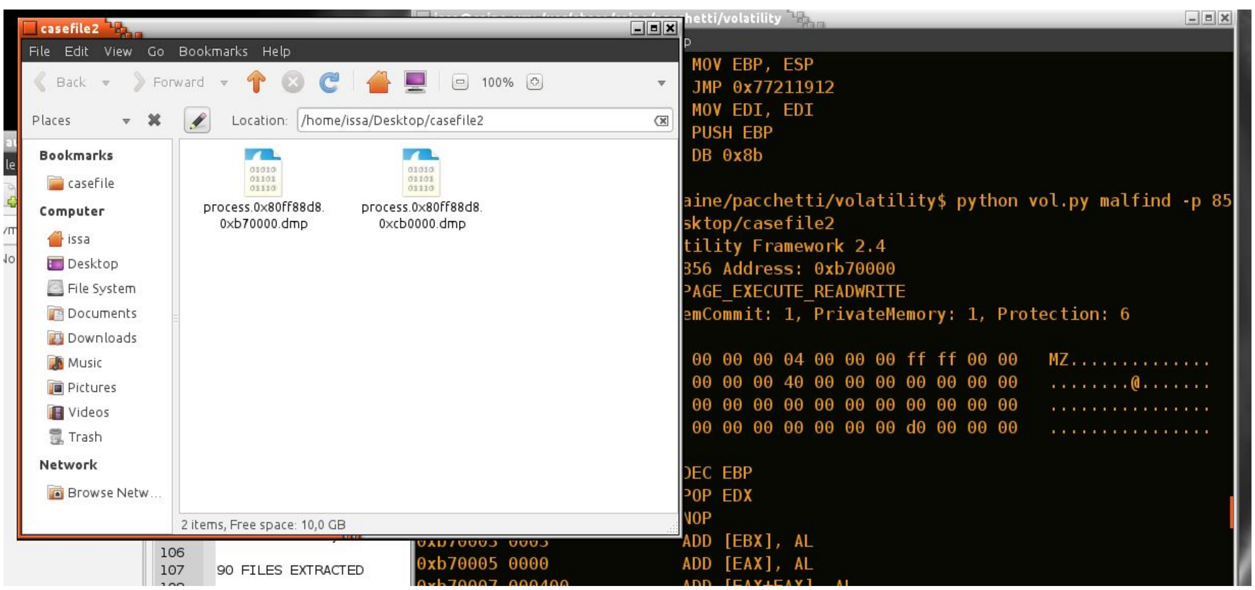
The report provides a listing of the extracted files. In this case, 90 files are extracted. The extracted files can be further analyzed by checking executable files against virus scanners or searching file contents for interesting information (e.g. using string search).

Using **malfind** plugin, allows finding injected code and dumping sections for specific PID. With it, all executables can be extracted from the processes running on the victim's machine. Each extracted file is named with its associated PID. We can also use the command to extract the executable from a specific process, as follows:

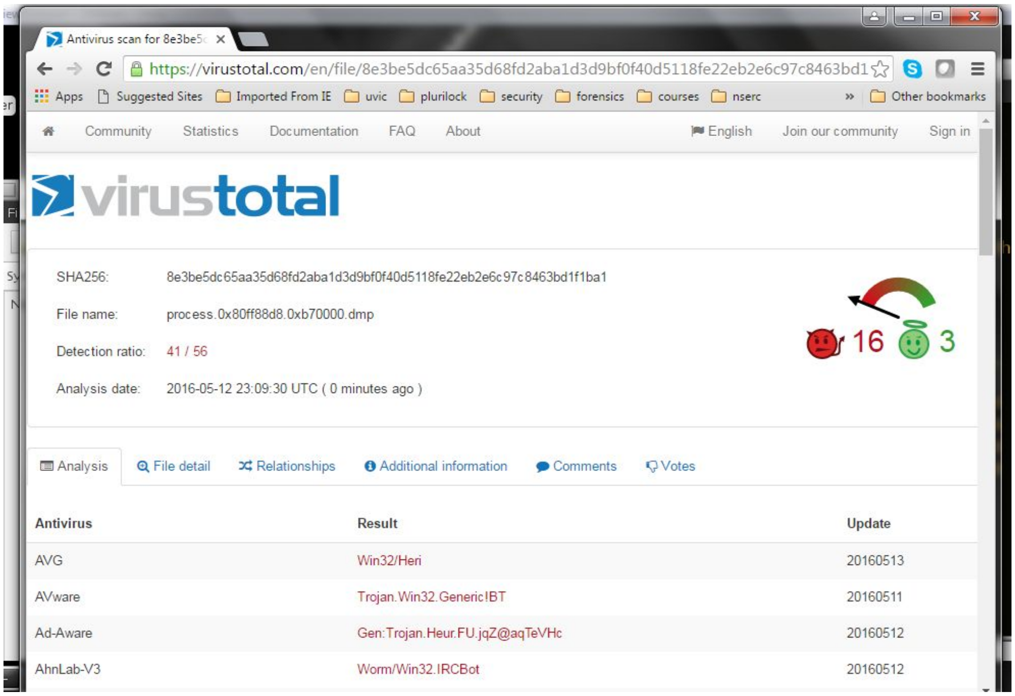
python vol.py malfind -p 856 --dump-dir /home/issa/Desktop/casefile2



The extracted files will be stored in the specified directory:



In this case, two files are generated. We can check those files by uploading them at an online virus scanner, e.g., virustotal.com:

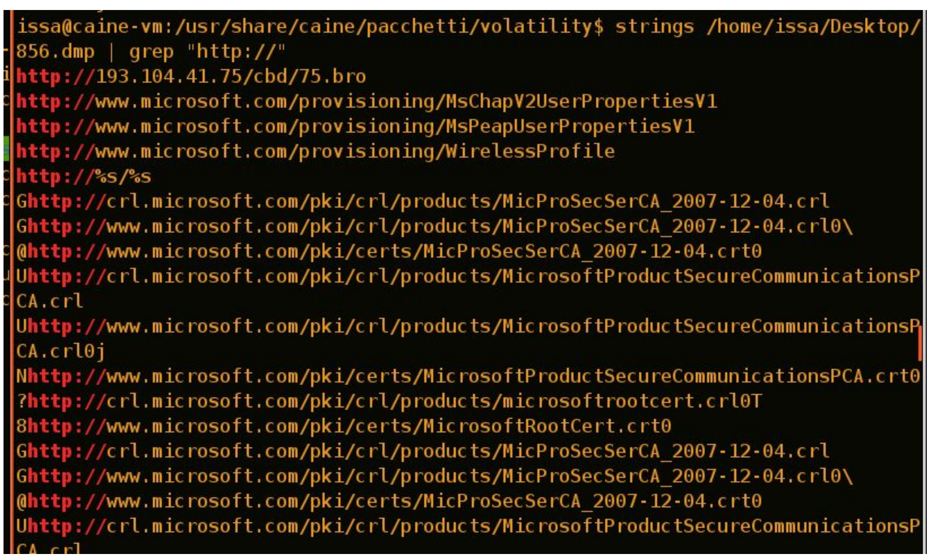


The results indicate that 41/56 scanners have classified the file as malware. Further online will yield more details on the suspected malware.

Additionally, we can search for any suspicious URLs that may be in the suspected process's memory, using string search as follows:

strings 856.dmp | grep "http://"

This returns several URLs as shown below:



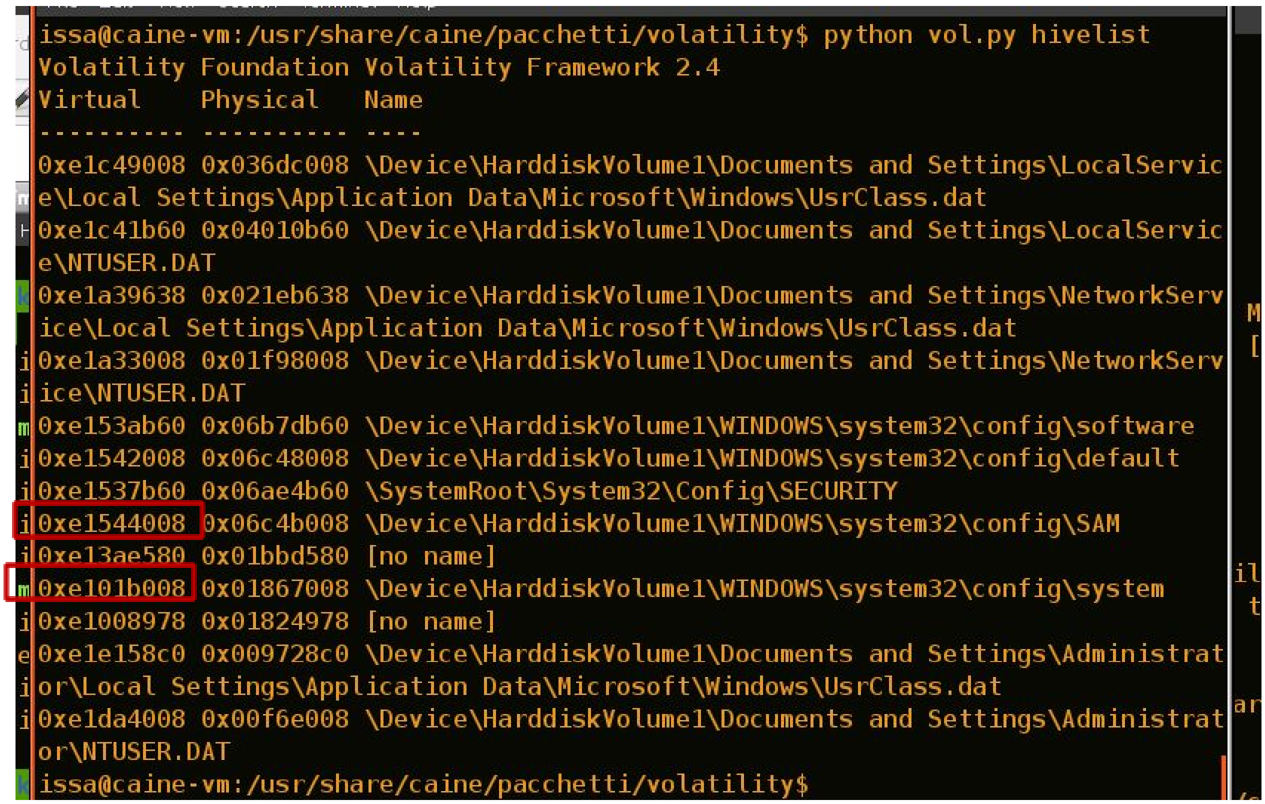
Further analysis of the visited sites can reveal valuable information for the investigation.

Registry Analysis

Registry analysis can also be useful in looking for sign of rootkits or code injection. Additionally, registry entries can reveal (if applicable) password hashes for accounts accessed.

The following command list the location in RAM of the Registry hives:

python vol.py hivelist



By reviewing the output, we can find the following critical two addresses outlined in red above the virtual addresses of the SAM and SYSTEM hives. Those two hives together contain enough information to extract Windows password hashes.

The password hashes can be extracted using the hashdump plugin as follows:

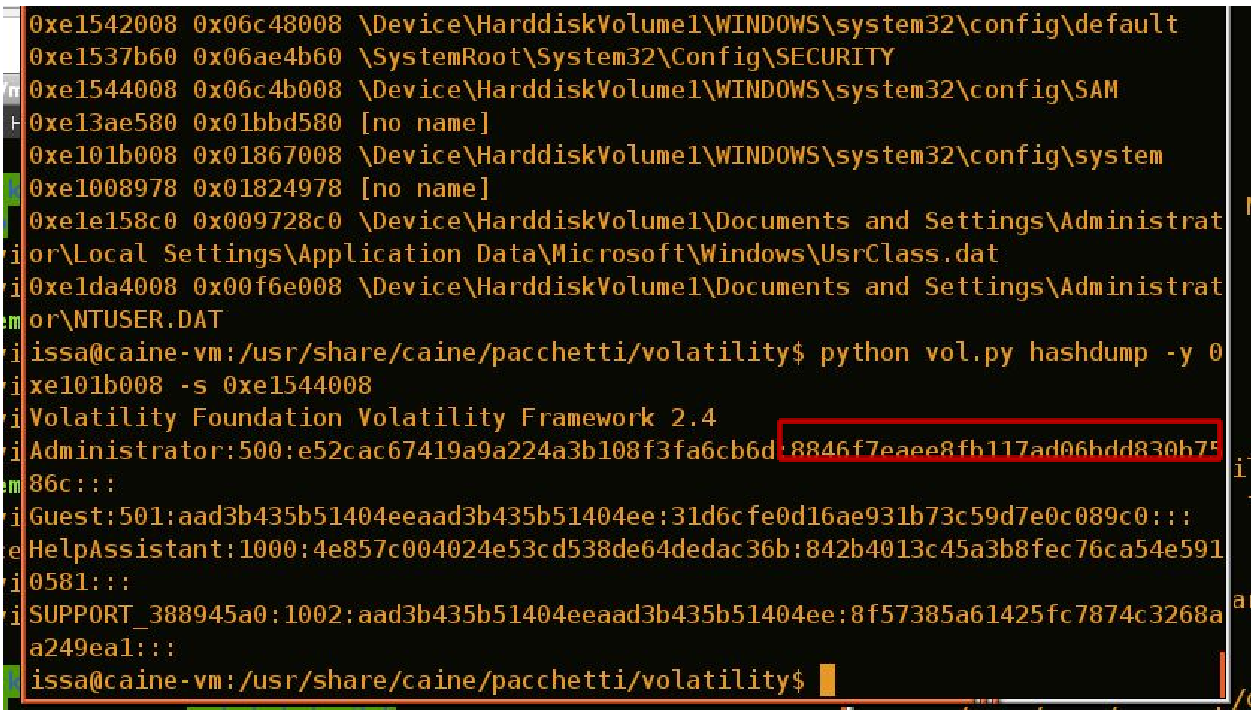
python vol.py hashdump -y 0xe101b008 -s 0xe1544008

Where the flags -y and -s point to the virtual addresses for the System and SAM hives, respectively.

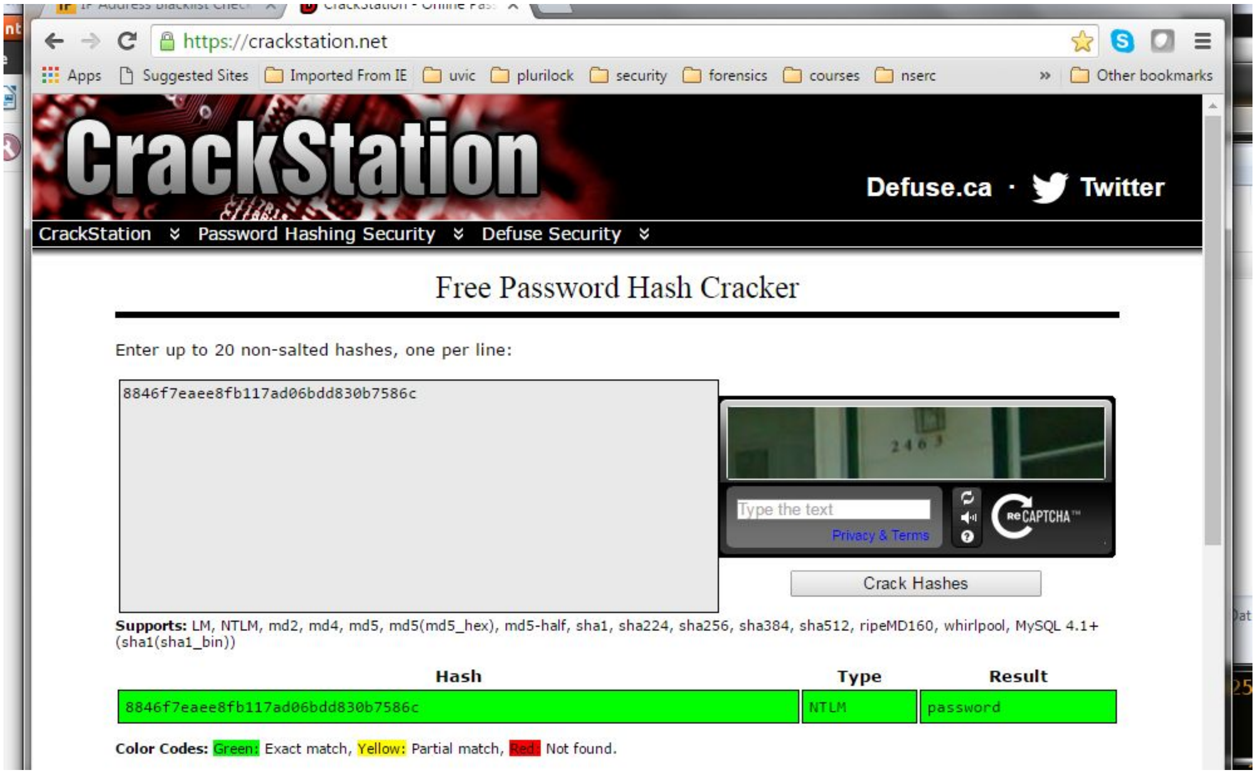
The output of the command are login account names and hashed passwords of the accounts accessed as shown below.

Windows stores two hashes with each password, delimited by colons. The first one is an extremely insecure, obsolete hash using the LANMAN algorithm. Windows operating systems since Vista no longer use LANMAN hashes, so they are filled with a dummy value starting with "aad".

The second hash is the newer NTLM hash, which is much better than LANMAN hashes, but still extremely insecure and much more easily cracked than Linux or Mac OS X hashes.



You can copy the hashes and submit them to a password cracking tool to extract the plaintext passwords. For instance, by running the hash for account Administrator outlined in green, against an online scanner, we obtain the plaintext password as the string “password”.



Summary

This tutorial illustrates forensics memory analysis using Volatility. Volatility is a powerful tool which relies on a host of plugins to enable in-depth study of memory dumps. Unfortunately, the current version is still heavily geared towards Windows dumps. The support for Linux dumps is still very sketchy.