TryHackMe: Critical

The following is a writeup for the Critical room hosted on TryHackMe. This room involves using Volatility to investigate a memory dump of a suspected compromised machine. It guides you through conducting basic investigations using Volatility, and I found it very useful in combination with the volatility room. It is beginner friendly so I highly recommend completing it.

Scenario: Our user “Hattori” has reported strange behaviour on his computer and realised that some PDF files have been encrypted, including a critical document to the company named important_document.pdf. He decided to report it; since it was suspected that some credentials might have been stole, the DFIR team has been involved and has captured some evidence. Join the team to investigate and learn how to get information from a memory dump in a practical scenario.

1. Memory forensics

Memory forensics is a subset of computer forensics that involves analysing volatile memory. In Windows, it involving analysing RAM which is volatile memory, meaning that its contents get flushed (wiped) upon reboot or shutdown. You can divide the tasks in a memory forensic investigation into two main phases: memory acquisition and memory analysis. During the memory acquisition phase, you copy the contents of memory to a file (this is called a dump). You then analyse this memory during the memory analysis phase.

2. Environment

In this room, Volatility has been installed and an alias was created. To run volatility, enter vol:

```
analyst@ip-10-10-72-197:~$ vol windows --help
Volatility 3 Framework 2.5.2
usage: volatility [-h] [-c CONFIG] [--parallelism [{processes,threads,off}]] [-e EXTEND]
                  [-p PLUGIN DIRS] [-s SYMBOL DIRS] [-v] [-l LOG] [-o OUTPUT DIR] [-q]
                  [-r RENDERER] [-f FILE] [--write-config] [--save-config SAVE CONFIG]
                  [--clear-cache] [--cache-path CACHE PATH] [--offline]
                  [--single-location SINGLE LOCATION] [--stackers [STACKERS [STACKERS ...]]]
                  [--single-swap-locations [SINGLE SWAP LOCATIONS [SINGLE SWAP LOCATIONS ...]]
]
                  plugin ...
volatility: error: argument plugin: plugin windows matches multiple plugins (windows.bigpools.
BigPools, windows.cachedump.Cachedump, windows.callbacks.Callbacks, windows.cmdline.CmdLine, w
indows.crashinfo.Crashinfo, windows.devicetree.DeviceTree, windows.dlllist.DllList, windows.dr
iverirp.DriverIrp, windows.drivermodule.DriverModule, windows.driverscan.DriverScan, windows.d
umpfiles.DumpFiles, windows.envars.Envars, windows.filescan.FileScan, windows.getservicesids.G
etServiceSIDs, windows.getsids.GetSIDs, windows.handles.Handles, windows.hashdump.Hashdump, wi
ndows.info.Info, windows.joblinks.JobLinks, windows.ldrmodules.LdrModules, windows.lsadump.Lsa
dump, windows.malfind.Malfind, windows.mbrscan.MBRScan, windows.memmap.Memmap, windows.mftscan
.ADS, windows.mftscan.MFTScan, windows.modscan.ModScan, windows.modules.Modules, windows.mutan
tscan.MutantScan, windows.netscan.NetScan, windows.netstat.NetStat, windows.poolscanner.PoolSc
anner, windows.privileges.Privs, windows.pslist.PsList, windows.psscan.PsScan, windows.pstree.
```

Some relevant plugins can be seen in the table below:

Windows.cmdline	Lists process command line arguments
windows.drivermodule	Determines if any loaded drivers were hidden by a rootkit
Windows.filescan	Scans for file objects present in a particular Windows memory image
Windows.getsids	Print the SIDs owning each process
Windows.handles	Lists process open handles
Windows.info	Show OS & kernel details of the memory sample being analyzed
Windows.netscan	Scans for network objects present in a particular Windows memory image
Widnows.netstat	Traverses network tracking structures present in a particular Windows memory image.
Windows.mftscan	Scans for Alternate Data Stream
Windows.pslist	Lists the processes present in a particular Windows memory image
Windows.pstree	List processes in a tree based on their parent process ID

3. Gathering target information

Gathering information about the target is crucial, it is very important to understand the specific architecture and OS in use (aka the OS and architecture of the target memory dump). You can get information about the target using the -f switch to indicate the file to analyse followed by the windows.info plugin:

```
vol -f memdump.mem windows.info
```

```
Variable        Value

Kernel Base     0xf8066161b000
DTB     0x1ad000
Symbols file:///home/analyst/volatility3-2.5.2/volatility3/symbols/windows/ntkrnlmp.pdb/4DBE14
4182FF4156845CD3BD8B654E56-1.json.xz
Is64Bit True
IsPAE   False
layer name      0 WindowsIntel32e
memory layer    1 FileLayer
KdVersionBlock  0xf8066222a400
Major/Minor     15.19041
MachineType     34404
KeNumberProcessors      2
SystemTime      2024-02-24 22:52:52
NtSystemRoot    C:\Windows
NtProductType   NtProductWinNt
NtMajorVersion  10
NtMinorVersion  0
PE MajorOperatingSystemVersion  10
PE MinorOperatingSystemVersion  0
PE Machine      34404
PE TimeDateStamp        Sat Jan 13 03:45:32 2085
```

Is the architecture of the machine x64 (64 bit)?

Y:

```
Is64Bit True
```

What is the version of the Windows OS?

10:

```
NtMajorVersion  10
```

What is the base address of the kernel?

0xf8066161b000:

```
Kernel Base     0xf8066161b000
```

4. Searching for suspicious activity

We can start trying to find suspicious activity by investigating network activity. To do this, you can use the windows.netstat plugin to see if there is an interesting or unusual connection:

```
vol -f memdump.mem windows.netstat
```

```
Offset  Proto   LocalAddr       LocalPort       ForeignAddr     ForeignPort     State   PID     Owner   Created

0xe50ed9170ac0  TCPv4   192.168.182.139 49723   192.16.49.85    80      CLOSE WAIT      4780    SearchApp.exe   2024-02-24 22:48:49.000000
0xe50ed8a4ca20  TCPv4   192.168.182.139 49814   52.142.223.178  443     SYN SENT        368     svchost.exe     2024-02-24 22:52:43.000000
0xe50ed9275a20  TCPv4   192.168.182.139 3389    192.168.182.150 49253   ESTABLISHED     744     svchost.exe     2024-02-24 22:47:52.000000
0xe50ed9df3a20  TCPv4   192.168.182.139 49745   13.107.213.254  443     CLOSE WAIT      4780    SearchApp.exe   2024-02-24 22:50:42.000000
0xe50ed8c52a20  TCPv4   192.168.182.139 49719   23.222.237.202  443     CLOSE WAIT      4780    SearchApp.exe   2024-02-24 22:48:47.000000
0xe50ed9427a20  TCPv4   192.168.182.139 49694   20.7.1.246      443     ESTABLISHED     368     svchost.exe     2024-02-24 22:47:54.000000
0xe50ed83ea4d0  TCPv4   192.168.182.139 49743   23.222.237.203  443     CLOSE WAIT      4780    SearchApp.exe   2024-02-24 22:50:39.000000
0xe50edac57a20  TCPv4   192.168.182.139 49712   152.199.55.200  443     CLOSE WAIT      4780    SearchApp.exe   2024-02-24 22:48:06.000000
0xe50ed9508a20  TCPv4   192.168.182.139 49744   23.222.237.203  443     CLOSE WAIT      4780    SearchApp.exe   2024-02-24 22:50:39.000000
0xe50ed6390cb0  TCPv4   0.0.0.0 135     0.0.0.0 0       LISTENING       896     svchost.exe     2024-02-24 22:47:36.000000
0xe50ed6390cb0  TCPv6   ::      135     ::      0       LISTENING       896     svchost.exe     2024-02-24 22:47:36.000000
0xe50ed6391910  TCPv4   0.0.0.0 135     0.0.0.0 0       LISTENING       896     svchost.exe     2024-02-24 22:47:36.000000
0xe50ed6391bd0  TCPv4   192.168.182.139 139     0.0.0.0 0       LISTENING       4       System  2024-02-24 22:47:36.000000
0xe50ed818d310  TCPv4   0.0.0.0 445     0.0.0.0 0       LISTENING       4       System  2024-02-24 22:47:37.000000
0xe50ed818d310  TCPv6   ::      445     ::      0       LISTENING       4       System  2024-02-24 22:47:37.000000
```

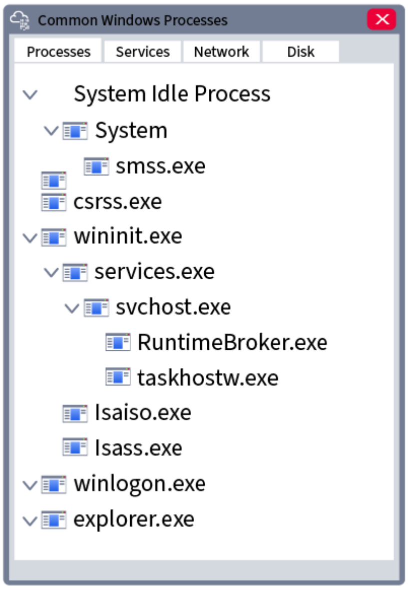
To start investigating running processes, you can use the windows.pstree plugin, which will display a tree of the processes running on the machine:

```
vol -f memdump.mem windows.pstree
```

```
PID     PPID    ImageFileName   Offset(V)       Threads Handles SessionId       Wow64   CreateTime      ExitTime

4       0       System  0xe50ed3687040  150     -       N/A     False   2024-02-24 22:47:35.000000      N/A
* 312   4       smss.exe        0xe50ed68b0040  2       -       N/A     False   2024-02-24 22:47:35.000000      N/A
* 1600  4       MemCompression  0xe50ed379e280  50      -       N/A     False   2024-02-24 22:47:36.000000      N/A
* 92    4       Registry        0xe50ed36ed080  4       -       N/A     False   2024-02-24 22:47:31.000000      N/A
424     400     csrss.exe       0xe50ed67d7140  9       -       0       False   2024-02-24 22:47:35.000000      N/A
500     400     wininit.exe     0xe50ed7366080  2       -       0       False   2024-02-24 22:47:35.000000      N/A
* 664   500     lsass.exe       0xe50ed7360080  8       -       0       False   2024-02-24 22:47:35.000000      N/A
* 776   500     fontdrvhost.ex  0xe50ed7c69140  6       -       0       False   2024-02-24 22:47:35.000000      N/A
* 636   500     services.exe    0xe50ed73d3080  6       -       0       False   2024-02-24 22:47:35.000000      N/A
** 896  636     svchost.exe     0xe50ed7d112c0  9       -       0       False   2024-02-24 22:47:36.000000      N/A
** 1924 636     svchost.exe     0xe50ed73ab2c0  5       -       0       False   2024-02-24 22:47:36.000000      N/A
** 3464 636     svchost.exe     0xe50ed88e3080  7       -       1       False   2024-02-24 22:47:39.000000      N/A
** 7312 636     SecurityHealth  0xe50ed9af1280  10      -       0       False   2024-02-24 22:47:56.000000      N/A
** 2964 636     dllhost.exe     0xe50ed858d280  14      -       0       False   2024-02-24 22:47:37.000000      N/A
** 3348 636     svchost.exe     0xe50ed8b722c0  6       -       0       False   2024-02-24 22:47:39.000000      N/A
** 7060 636     WUDFHost.exe    0xe50ed9ad41c0  9       -       0       False   2024-02-24 22:47:53.000000      N/A
** 792  636     svchost.exe     0xe50ed7c85240  13      -       0       False   2024-02-24 22:47:35.000000      N/A
*** 3336        792     LockApp.exe     0xe50ed99b70c0  19      -       1       False   2024-02-24 22:47:46.000000      N/A
*** 5008        792     RuntimeBroker.  0xe50ed916c300  12      -       1       False   2024-02-24 22:47:40.000000      N/A
*** 4368        792     RuntimeBroker.  0xe50edab30080  5       -       1       False   2024-02-24 22:49:46.000000      N/A
*** 4500        792     RuntimeBroker.  0xe50ed883f0c0  4       -       1       False   2024-02-24 22:47:40.000000      N/A
*** 6292        792     PhoneExperienc  0xe50ed99da080  17      -       1       False   2024-02-24 22:47:47.000000      N/A
*** 7188        792     smartscreen.ex  0xe50ed9e29300  7       -       1       False   2024-02-24 22:47:56.000000      N/A
*** 7452        792     WmiPrvSE.exe    0xe50eda204280  6       -       0       False   2024-02-24 22:47:57.000000      N/A
*** 5920        792     ShellExperienc  0xe50ed96c71c0  12      -       1       False   2024-02-24 22:47:45.000000      N/A
*** 2848        792     FileCoAuth.exe  0xe50ed95d2080  6       -       1       False   2024-02-24 22:52:05.000000      N/A
*** 4780        792     SearchApp.exe   0xe50ed9299080  47      -       1       False   2024-02-24 22:47:40.000000      N/A
```

It is difficult to identify suspicious processes as threat actors often try to disguise their malicious processes as legitimate ones. One way to find suspicious/malicious processes is to understand how legitimate processes (especially those related to the OS) are represented (like their child or parent processes, etc):



Using the plugin "windows.netscan" can you identify the IP address that established a connection on port 80?

You can use the following command to start answering this question:

```
vol -f memdump.mem windows.netscan
```

```
192.168.182.139 49817   192.168.182.128 80      ESTABLISHED     8300    msedge.exe      2024-02-24 22:52:53.000000
```

Using the plugin windows.netscan, can you identify the program (owner) used to access through port 80?

You can see the answer in the image form the previous answer, the program (owner) is msedge.exe.

Analysing the process present on the dump, what is the PID of the child process of critical_updat?

We can use the windows.pstree plugin to find the associated PID of the child process critical_updat:

```
vol -f memdump.mem windows.pstree
```

```
**** 1648        7960    critical updat
***** 1612       1648    updater.exe
```

As you can see in the above image, 1612 is the PID of the child process.

What is the time stamp for the process with the truncated name critical_updat?

You can find the timestamp in the output of the previous question:

```
critical_updat  0xe50ed94c1080  5       -       1       False   2024-02-24 22:51:50.000000
```

5. Finding interesting data

Let's start investigating the critical_updat process that has a child process called updater. Stating off, let's looking into the child process by seeing where it was saved on disk. To do this, you can use the windows.filescan plugin. The output is large, so save it to a file using '>':

```
vol -f memdump.mem windows.filescan > filescan_out
```

Let's now examine the output using cat and piping it to grep:

```
analyst@ip-10-10-72-197:~$ cat filescan_out | grep updater
0xe50ed736e8a0  \Users\user01\Documents\updater.exe     216
0xe50ed846fc60  \Program Files (x86)\Microsoft\EdgeUpdate\1.3.185.17\msedgeupdateres_en.dll     216
0xe50ed8482d10  \Program Files (x86)\Microsoft\EdgeUpdate\1.3.185.17\msedgeupdateres_en.dll     216
```

If you want even more detailed information such as when the file was accessed or modified, you can use the windows.mftscan.MFTScan plugin:

```
vol -f memdump.mem windows.mftscan.MFTScan > mftout
```

```
analyst@ip-10-10-72-197:~$ cat mftout | grep updater
* 0xd389c63ce528        FILE    111417  2       File    Archive FILE_NAME       2024-02-24 22:51:50.000000      2024-02-24 22:51:50.000000      2024-02-24 22:51:50.000000      2024-02-24 22:
51:50.000000    updater[1].exe
```

Getting the goods

Let's now dump the memory region corresponding to updater.exe and examine it. To do so, we can use the windows.memmap plugin. This time, we will specify the output directory with the -o switch (we will use the same directory which is denoted by ".") and the option –dump followed by the option –pid and the PID of the process, which in the case of updater.exe is 1612:

```
vol -f memdump.mem -o . windows.memmap --dump --pid 1612
```

Once the command has finished running, you will have a file with an extension of .dmp. Let's use the strings command to view the .dmp file:

```
strings pid.1612.dmp |less
```

```
PROCESSOR_IDENTIFIER=AMD64 Family 25 Model 97 Stepping 2, AuthenticAMD
hZG_
tN}frL
tN}frL
CommonProgramFiles(x86)=C:\Program Files (x86)\Common Files
PATHEXT=.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
h8H$
DriverData=C:\Windows\System32\Drivers\DriverData
8[G_
USERDOMAIN_ROAMINGPROFILE=DESKTOP-3NMNM0H
C:\Users\user01\Documents\updater.exe
WB0_
SB<_
_B8_
http://key.critical-update.com/encKEY.txt
LOCALAPPDATA=C:\Users\user01\AppData\Local
CommonProgramFiles=C:\Program Files\Common Files
ProgramFiles(x86)=C:\Program Files (x86)
```

```
HDcl
=}>a
h9cl
csvc
`Bcl
HDcl
important_document.pdf
s\user01\Documents
x@cl
HDcl
X&=l
(YDj
(ZDj
```

You can also use grep to search for things like HTTP, file extensions, etc. The -B and -A switches can be set to 10 to look for 10 lines above and below the match:

```
strings pid.1612.dmp |grep -B 10 -A 10 "http://key.critical-update.com/encKEY.txt"
```

Analysing the "windows.filescan" output, what is the full path and name for critical_updat?

To answer this, you use the windows.filescan plugin. The output is large, so save it to a file using '>':

```
vol -f memdump.mem windows.filescan > filescan out
```

Let's now examine the output using cat and piping it to grep:

```
analyst@ip-10-10-72-197:~$ cat filescan out | grep updater
0xe50ed736e8a0  \Users\user01\Documents\updater.exe     216
0xe50ed846fc60  \Program Files (x86)\Microsoft\EdgeUpdate\1.3.185.17\msedgeupdateres en.dll      216
0xe50ed8482d10  \Program Files (x86)\Microsoft\EdgeUpdate\1.3.185.17\msedgeupdateres en.dll      216
```

The full path is therefore C:\Users\user01\Documents\critical_update.exe.

Analysing the “windows.mftscan.MFTScan” what is the Timestamp for the created data of important_document.pdf?

Run the following command:

```
vol -f memdump.mem windows.mftscan.MFTScan > mftout
```

We can now use grep to find the important_document.pdf file:

```
analyst@ip-10-10-72-197:~$ cat mftout | grep important_document.pdf
* 0xd389c5fbad28        FILE    111083  2       File    Archive FILE NAME       2024-02-24 20:39:42.000000      2024-02-24 20:39:42.000000      2024-02-24 20:39:42.000000      2024-02-24 20:
39:42.000000    important_document.pdf
```

The answer is 2024-02-24 20:39:42.000000.

Analysing the updater.exe memory output, can you observe the HTTP request and determine the server used by the attacker?

Enter the following command which dumps the memory region corresponding to the malicious binary we identified earlier (updater.exe):

```
vol -f memdump.mem -o . windows.memmap --dump --pid 1612
```

Once the command has finished running, you will have a file with an extension of .dmp. Let’s use the strings command to view the .dmp file and pipe it to grep to look for “HTTP”:

```
strings pid.1612.dmp | grep "http://" | head -n 10
```

```
http://key.critical-update.com/encKEY.txt
http://key.critical-update.com/encKEY.txt
http://key.critical-update.com/encKEY.txt
http://key.critical-update.com/encKEY.txt
```

Here we can see some interesting domains, so let’s grep for this:

```
strings pid.1612.dmp | grep -B 10 -A 10 "http://key.critical-update.com/encKEY.txt"
```

```
http://key.critical-update.com/encKEY.txt
HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/3.10.4
```

In the above image, we can see a request header which contains the server type which is:

SimpleHTTP/0.6 Python/3.10.4