# TryHackMe— Volatility Room Practical Challenge Walkthrough

## Endpoint Investigation with Volatility 3

Drew Arpino · Follow
15 min read · Feb 26, 2024

Listen Share More

Image Credit: https://github.com/volatilityfoundation

### Introduction:

Hello! Last week's write-up was for the **LetsDefend** Memory Analysis room which was my introduction to the *Volatility* framework. This week, I am going to build on my knowledge and am writing up my learning with the excellent *Volatility* room on **TryHackMe**. The capstone of the room is a practical challenge with two cases.

**TryHackMe** makes challenges like these very beginner-friendly and the coursework modules prior to the challenge will have you well-prepared. This challenge does require some additional, external research but it definitely helps to add context and spend more time on the DFIR process. In the spirit of learning and research I am not going to reveal the flags this time around but I will walk you through my process so you can recreate it yourself.

I used *Volatility 3* to complete this room but going forward I will use the terms *Volatility 3* and *Volatility* interchangeably. This is a longer one, so get comfortable. Thanks for reading!

**Challenge Link:** https://tryhackme.com/room/volatility

**Challenge Scenarios:**

> ***Case 001 — BOB! THIS ISN'T A HORSE!***
>
> *Your SOC has informed you that they have gathered a memory dump from a quarantined endpoint thought to have been compromised by a banking trojan masquerading as an Adobe document. Your job is to use your knowledge of threat intelligence and reverse engineering to perform memory forensics on the infected host.*
>
> *You have been informed of a suspicious IP in connection to the file that could be helpful.* `41[.]168[.]5[.]140`
>
> *The memory file is located in* `/Scenarios/Investigations/Investigation-1.vmem`
>
> ***Case 002 — That Kind of Hurt my Feelings***
>
> *You have been informed that your corporation has been hit with a chain of ransomware that has been hitting corporations internationally. Your team has already retrieved the decryption key and recovered from the attack. Still, your job is to perform post-incident analysis and identify what actors were at play and what occurred on your systems. You have been provided with a raw memory dump from your team to begin your analysis.*
>
> *The memory file is located in* `/Scenarios/Investigations/Investigation-2.raw`

## Case 001 — BOB! THIS ISN'T A HORSE!

## Questions 1 & 2:

### What is the build version of the host machine in Case 001?

### At what time was the memory file acquired in Case 001?

Before we get started, I want to call out the *Volatility 3* help command built into the tool. We're going to lean on this a lot. This is a great way to explore what plugins are available and get a brief description of their functions. In some cases, the plugin itself may have its own set of help for optional arguments! Don't worry, we will utilize these further in the challenge. For now, I will leave the help command here as a starting point if you'd prefer to navigate the challenge on your own.

```
python3 vol.py -h
```

Okay, let's get started! While the challenge doesn't specify it, I am going to assume that we are analyzing a memory dump from a *Windows* endpoint. If you have completed the preceding tasks already in the TryHackMe *Volatility* room, you will have come across a module that will help us get started with scoping the challenge and working through the case: **windows.info**

```
windows.info.Info   Show OS & kernel details of the memory sample being analyzed.
```

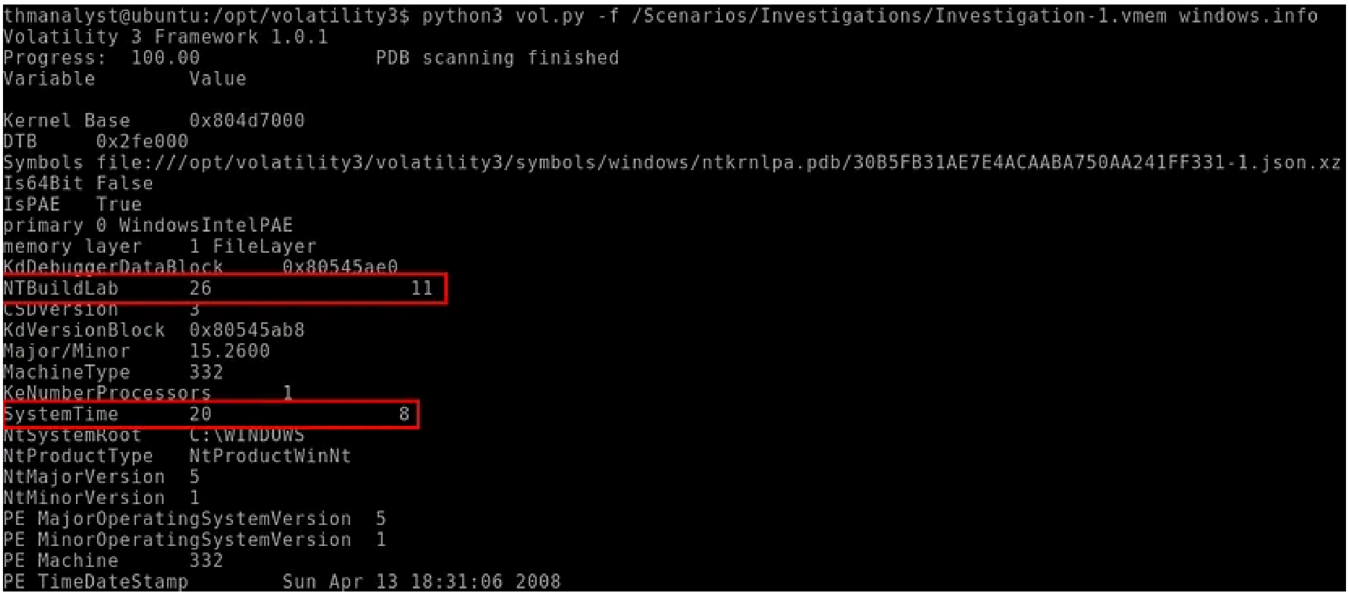
As a refresher, Task 6 states:

> *If we are still looking to get information about what the host is running from the memory dump, we can use the following three plugins* `windows.info` `linux.info` `mac.info`*. This*

high-level details from the dump file and better understand our victim environment. When we run *Volatility* we'll point to the challenge file path with the -f parameter and have it use the **windows.info** plugin.

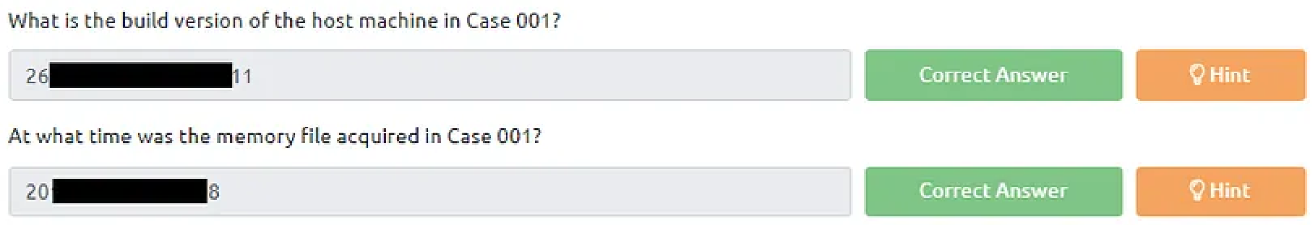
```
python3 vol.py -f /Scenarios/Investigations/Investigation-1.vmem windows.info
```

Once *Volatility* does its magic, we get the following output with some details of the memory dump.



```
thmanalyst@ubuntu:/opt/volatility3$ python3 vol.py -f /Scenarios/Investigations/Investigation-1.vmem windows.info
Volatility 3 Framework 1.0.1
Progress:  100.00               PDB scanning finished
Variable        Value

Kernel Base     0x804d7000
DTB     0x2fe000
Symbols file:///opt/volatility3/volatility3/symbols/windows/ntkrnlpa.pdb/30B5FB31AE7E4ACAABA750AA241FF331-1.json.xz
Is64Bit False
IsPAE   True
primary 0 WindowsIntelPAE
memory layer    1 FileLayer
KdDebuggerDataBlock     0x80545ae0
NTBuildLab      26 [redacted] 11
CSDVersion      3
KdVersionBlock  0x80545ab8
Major/Minor     15.2600
MachineType     332
KeNumberProcessors      1
SystemTime      20 [redacted] 8
NtSystemRoot    C:\WINDOWS
NtProductType   NtProductWinNt
NtMajorVersion  5
NtMinorVersion  1
PE MajorOperatingSystemVersion  5
PE MinorOperatingSystemVersion  1
PE Machine      332
PE TimeDateStamp        Sun Apr 13 18:31:06 2008
```

I think the *NTBuildLab & SystemTime* fields *should* answer **questions 1 & 2** — let's submit to confirm that we have the right answers:



What is the build version of the host machine in Case 001?

26[redacted]11

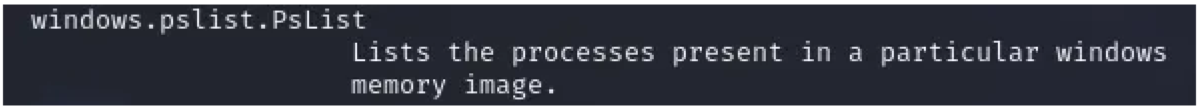
At what time was the memory file acquired in Case 001?

20[redacted]8

## Question 3:

### What process can be considered suspicious in Case 001?

Okay, now let's get into the analysis and use *Volatility* to dig a bit deeper and understand the running processes on the victim system at the time the memory dump was taken. If we refer to the *Volatility* help again we have several process identification options.

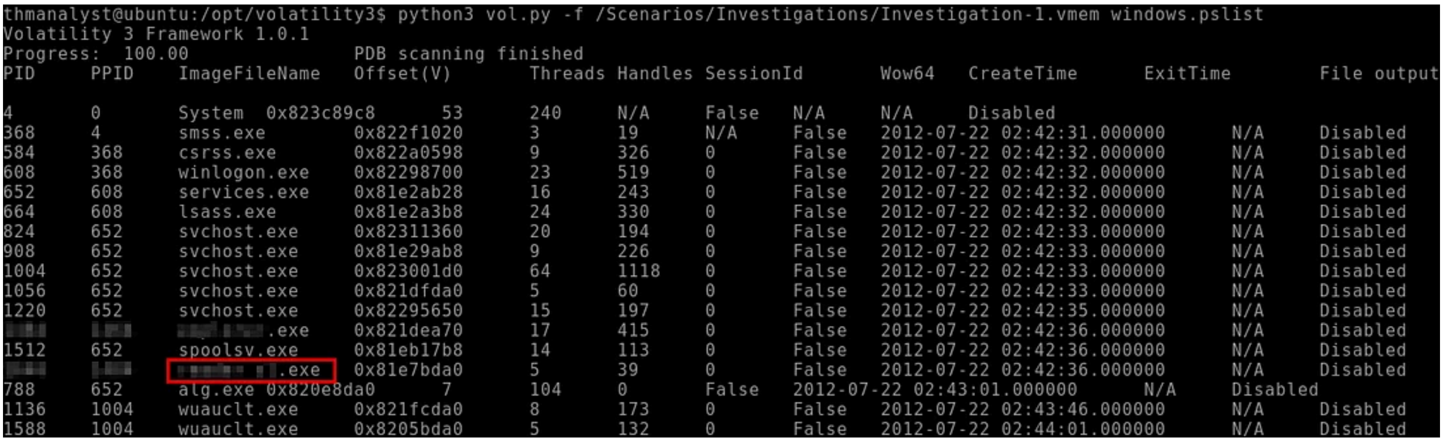


```
windows.pslist.PsList
                    Lists the processes present in a particular windows
                    memory image.
```

The windows.pslist help file entry.

Let's go with the light-touch option first and simply list out the processes list using the **windows.pslist** plugin. We'll see if we can find anything suspicious within our case file.

```
python3 vol.py -f /Scenarios/Investigations/Investigation-1.vmem windows.pslist
```



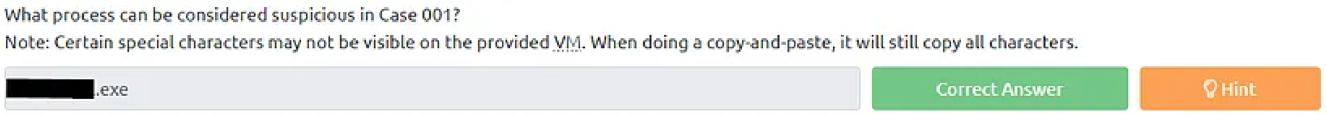
```
thmanalyst@ubuntu:/opt/volatility3$ python3 vol.py -f /Scenarios/Investigations/Investigation-1.vmem windows.pslist
Volatility 3 Framework 1.0.1
Progress:  100.00               PDB scanning finished
PID     PPID    ImageFileName   Offset(V)       Threads Handles SessionId       Wow64   CreateTime      ExitTime        File output

4       0       System  0x823c89c8      53      240     N/A     False   N/A     N/A     Disabled
368     4       smss.exe        0x822f1020      3       19      N/A     False   2012-07-22 02:42:31.000000      N/A     Disabled
584     368     csrss.exe       0x822a0598      9       326     0       False   2012-07-22 02:42:32.000000      N/A     Disabled
608     368     winlogon.exe    0x82298700      23      519     0       False   2012-07-22 02:42:32.000000      N/A     Disabled
652     608     services.exe    0x81e2ab28      16      243     0       False   2012-07-22 02:42:32.000000      N/A     Disabled
664     608     lsass.exe       0x81e2a3b8      24      330     0       False   2012-07-22 02:42:32.000000      N/A     Disabled
824     652     svchost.exe     0x82311360      20      194     0       False   2012-07-22 02:42:33.000000      N/A     Disabled
908     652     svchost.exe     0x81e29ab8      9       226     0       False   2012-07-22 02:42:33.000000      N/A     Disabled
1004    652     svchost.exe     0x823001d0      64      1118    0       False   2012-07-22 02:42:33.000000      N/A     Disabled
1056    652     svchost.exe     0x821dfda0      5       60      0       False   2012-07-22 02:42:33.000000      N/A     Disabled
1220    652     svchost.exe     0x82295650      15      197     0       False   2012-07-22 02:42:35.000000      N/A     Disabled
[redacted] [redacted] [redacted].exe 0x821dea70 17  415     0       False   2012-07-22 02:42:36.000000      N/A     Disabled
1512    652     spoolsv.exe     0x81eb17b8      14      113     0       False   2012-07-22 02:42:36.000000      N/A     Disabled
[redacted] [redacted] [redacted].exe 0x81e7bda0 5   39      0       False   2012-07-22 02:42:36.000000      N/A     Disabled
788     652     alg.exe 0x820e8da0      7       104     0       False   2012-07-22 02:43:01.000000      N/A     Disabled
1136    1004    wuauclt.exe     0x821fcda0      8       173     0       False   2012-07-22 02:43:46.000000      N/A     Disabled
1588    1004    wuauclt.exe     0x8205bda0      5       132     0       False   2012-07-22 02:44:01.000000      N/A     Disabled
```

Okay, now we have our output, see anything odd? I mentioned this in my previous *Volatility* post, but typically when looking at a *Windows* process list, I like to refer to the SANS Hunt Evil reference poster to understand normal *Windows* processes which helps tremendously during analysis.

Fortunately, this is a pretty short list and one of these process sticks out to me. Let's confirm our suspicion and submit the answer but before we do, pay attention to the note on the submissions page…

> *Note: Certain special characters may not be visible on the provided VM. When doing a copy-and-paste, it will still copy all characters.*

While we are here, let's make a special note to grab the process ID (PID) of the suspicious process as well, we will need this to answer Question 5. So now we have the PID as well, let's copy directly from the virtual machine, and paste our answer!



What process can be considered suspicious in Case 001?
Note: Certain special characters may not be visible on the provided VM. When doing a copy-and-paste, it will still copy all characters.

[redacted].exe

## Questions 4, 5, 6:

**What is the parent process of the suspicious process in Case 001?**

**What is the PID of the suspicious process in Case 001?**

**What is the parent process PID in Case 001?**

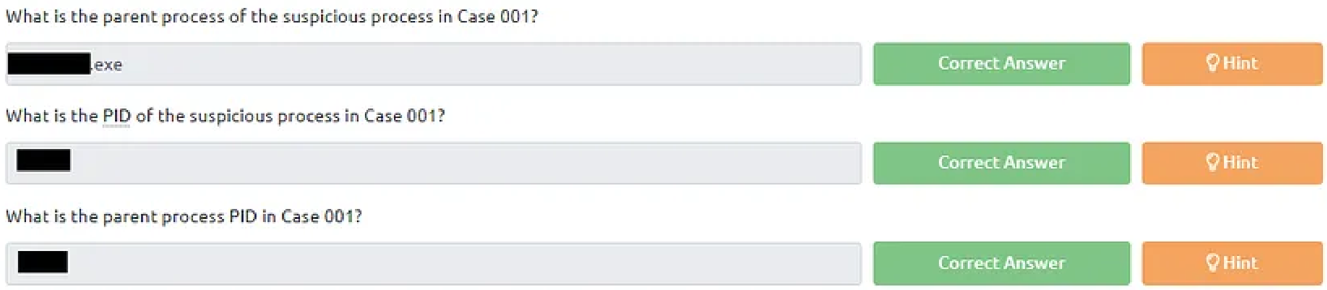
Good work! Now that we have located the suspicious process, these next few questions will be straightforward. We just need to look at the output of **pslist** and look at the information presented. These questions seem out of order to me but we'll figure it out.

Look at the columns in the output. We are going to focus on *Process ID (PID)*, *Parent Process ID (PPID)*, and *ImageFileName*. Using the information in these columns, we can determine the answers.

**Question 4** is looking for the *ImageFileName* of the parent process of the suspicious child process we located. To find it, search the **pslist** output and look at the *PPID* of the suspicious process (this could also answer **Question 6…**) Then, locate the process with the matching *PID* — this is the parent process and we can use the *ImageFileName* as our answer. Once you find it, make a note of the PID as well so we can have it ready for **Question 6**!

Remember in **Question 3** we made a note of the PID of the suspicious process? Now we can utilize it! **Question 5** is asking for the PID of the suspicious process — easy enough, we will simply use the PID value of the suspicious process for our answer.

Whew! We got them — let's move on.



What is the parent process of the suspicious process in Case 001?

[redacted].exe

What is the PID of the suspicious process in Case 001?

[redacted]

What is the parent process PID in Case 001?

[redacted]

## Questions 7 & 8:

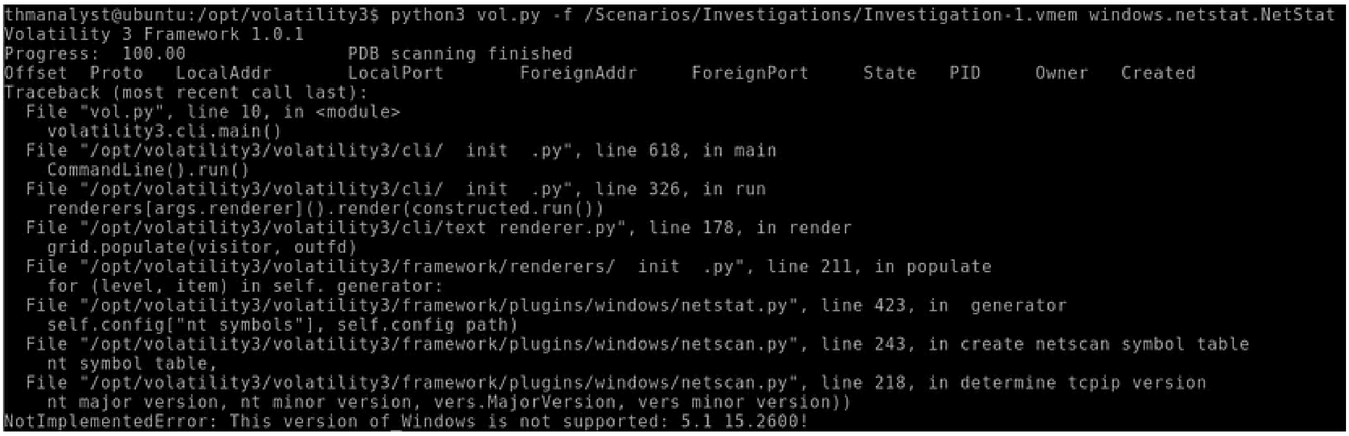
**What user-agent was employed by the adversary in Case 001?**

**Question 8: Was Chase Bank one of the suspicious bank domains found in Case 001? (Y/N)**

Cool, I haven't had a chance to look at the networking modules in *Volatility 3* yet. We'll start with the information given in the challenge scenario:

> *You have been informed of a suspicious IP in connection to the file that could be helpful.*
>
> `41.168.5.140`

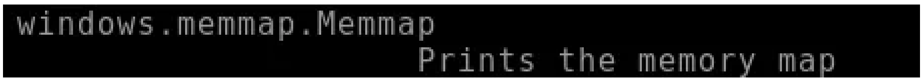
We have an IP, let’s see if we can get any networking info with **windows.netstat & windows.netscan.** Hmmm, the version of *Windows* our memory dump was taken from doesn’t seem to be supported…



```
thmanalyst@ubuntu:/opt/volatility3$ python3 vol.py -f /Scenarios/Investigations/Investigation-1.vmem windows.netstat.NetStat
Volatility 3 Framework 1.0.1
Progress:  100.00               PDB scanning finished
Offset  Proto   LocalAddr       LocalPort       ForeignAddr     ForeignPort     State   PID     Owner   Created
Traceback (most recent call last):
  File "vol.py", line 10, in <module>
    volatility3.cli.main()
  File "/opt/volatility3/volatility3/cli/  init  .py", line 618, in main
    CommandLine().run()
  File "/opt/volatility3/volatility3/cli/  init  .py", line 326, in run
    renderers[args.renderer]().render(constructed.run())
  File "/opt/volatility3/volatility3/cli/text renderer.py", line 178, in render
    grid.populate(visitor, outfd)
  File "/opt/volatility3/volatility3/framework/renderers/  init  .py", line 211, in populate
    for (level, item) in self. generator:
  File "/opt/volatility3/volatility3/framework/plugins/windows/netstat.py", line 423, in  generator
    self.config["nt symbols"], self.config path)
  File "/opt/volatility3/volatility3/framework/plugins/windows/netscan.py", line 243, in create netscan symbol table
    nt symbol table,
  File "/opt/volatility3/volatility3/framework/plugins/windows/netscan.py", line 218, in determine tcpip version
    nt major version, nt minor version, vers.MajorVersion, vers minor version))
NotImplementedError: This version of Windows is not supported: 5.1 15.2600!
```

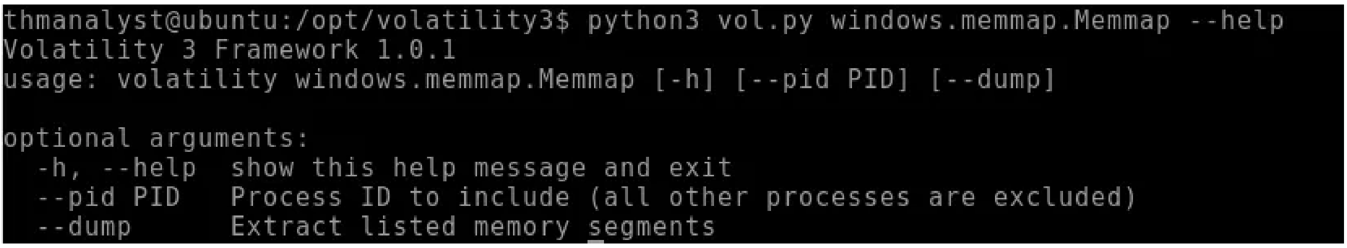
Let’s pivot and try something else. If we scan through the help files again, there isn’t an obvious plugin that can work to search for this suspicious IP address though…

What if we could dump out the suspicious processes’ memory map? Maybe we can get some additional information or perform further analysis about the contents of files opened by this process that are mapped to the memory address space…



```
windows.memmap.Memmap
                          Prints the memory map
```

Remember that before we started the investigation, I mentioned that some plugins have optional arguments? Here is a good example.



```
thmanalyst@ubuntu:/opt/volatility3$ python3 vol.py windows.memmap.Memmap --help
Volatility 3 Framework 1.0.1
usage: volatility windows.memmap.Memmap [-h] [--pid PID] [--dump]

optional arguments:
  -h, --help  show this help message and exit
  --pid PID   Process ID to include (all other processes are excluded)
  --dump      Extract listed memory segments
```

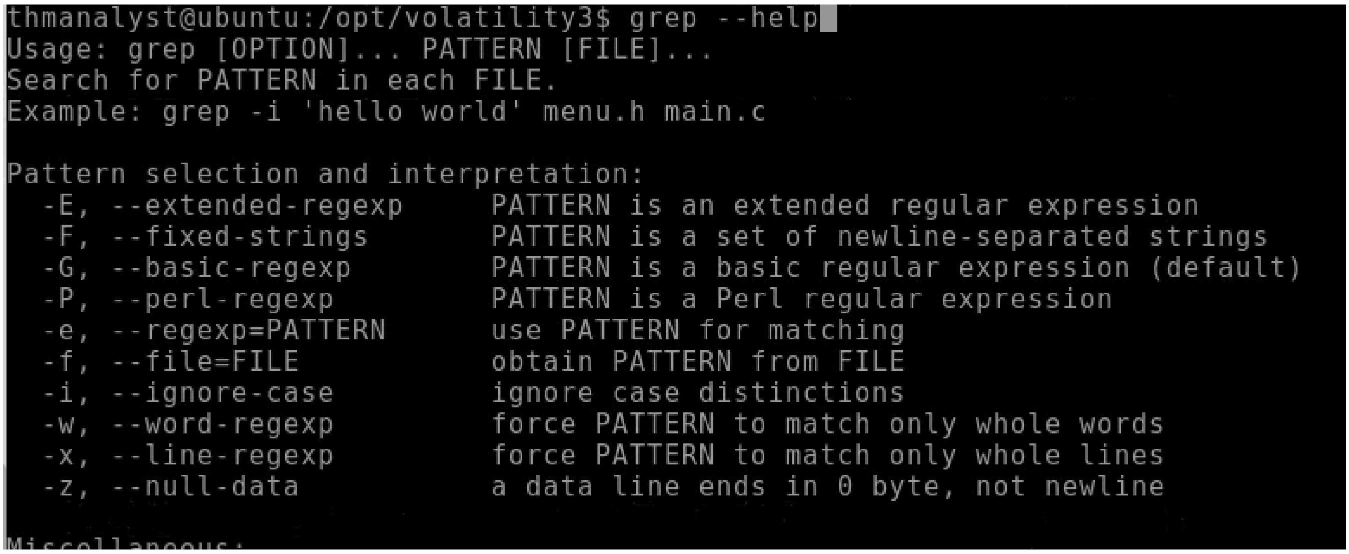
We see that the **memmap** plugin has some additional options that will help us here. We can try dumping the suspicious process that we identified earlier. This time we are going to set an output directory with the **-o** parameter:

```
python3 vol.py -f /Scenarios/Investigations/Investigation-1.vmem -o <output dir
```

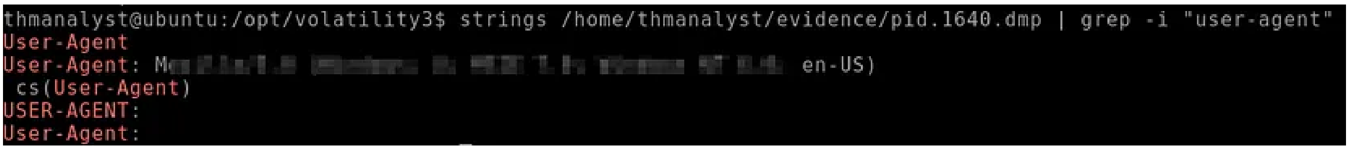
This creates a dump file which contains way too much information for us to manually sift through. Let’s try to utilize the **strings** command in Ubuntu and grep our output to be a bit more focused.

**Strings** is a command that searches the contents of a file for printable strings so it can help us pull out something human readable from the process dump.

So what are we going to grep? Well, if we read the question back, it asks for a *user-agent* so let’s just try that? If you aren’t familiar a user-agent headers are strings that servers use to identify requesting client details like the operating system or the web browser version. In this case, let’s use the **-i** argument to ignore case and just search for *user-agent.*



```
thmanalyst@ubuntu:/opt/volatility3$ grep --help
Usage: grep [OPTION]... PATTERN [FILE]...
Search for PATTERN in each FILE.
Example: grep -i 'hello world' menu.h main.c

Pattern selection and interpretation:
  -E, --extended-regexp     PATTERN is an extended regular expression
  -F, --fixed-strings       PATTERN is a set of newline-separated strings
  -G, --basic-regexp        PATTERN is a basic regular expression (default)
  -P, --perl-regexp         PATTERN is a Perl regular expression
  -e, --regexp=PATTERN      use PATTERN for matching
  -f, --file=FILE           obtain PATTERN from FILE
  -i, --ignore-case         ignore case distinctions
  -w, --word-regexp         force PATTERN to match only whole words
  -x, --line-regexp         force PATTERN to match only whole lines
  -z, --null-data           a data line ends in 0 byte, not newline

Miscellaneous:
```



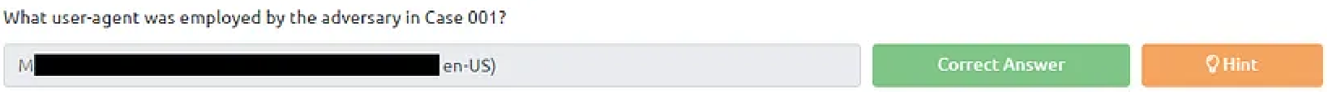
```
thmanalyst@ubuntu:/opt/volatility3$ strings /home/thmanalyst/evidence/pid.1640.dmp | grep -i "user-agent"
User-Agent
User-Agent: M[redacted] en-US)
 cs(User-Agent)
USER-AGENT:
User-Agent:
```

Awesome! It looks like we found something useful for our investigation that should answer **Question 7.**

Now let’s tackle **Question 8** and wrap Case 001 up. Since we already have the memory map for the suspicious process, maybe we can try the same logic as we did for **Question 7** and just grep out “Chase” — could that work? Try it and find out!

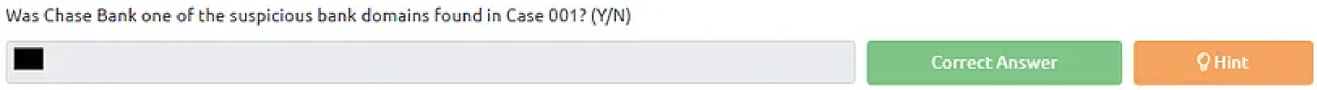
```
sudo strings /home/thmanalyst/evidence/pid.<redacted>.dmp | grep -i "Chase"
```

Great! Now we can submit, and close the case before moving on to our next set of challenges in Case 002!



What user-agent was employed by the adversary in Case 001?

M[redacted]en-US)



Was Chase Bank one of the suspicious bank domains found in Case 001? (Y/N)

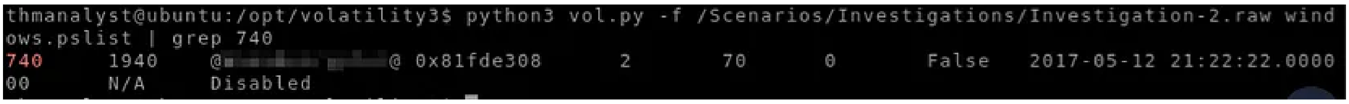
[redacted]

## Case 002 — That Kind of Hurt my Feelings

### Question 9: What suspicious process is running at PID 740 in Case 002?

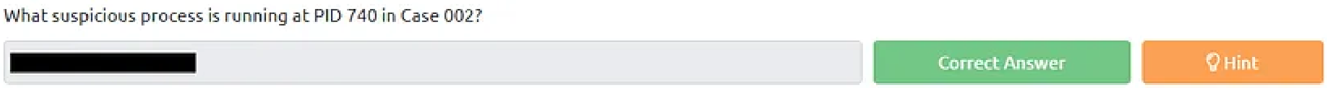
Okay! Case 002 is an analysis of a ransomware strain. Since we have the PID of the suspicious process already, let’s use the **pslist** plugin again and this time let’s grep only the suspicious PID:

```
python3 vol.py -f /Scenarios/Investigations/Investigation-2.raw wind
ows.pslist | grep 740
```



```
thmanalyst@ubuntu:/opt/volatility3$ python3 vol.py -f /Scenarios/Investigations/Investigation-2.raw wind
ows.pslist | grep 740
740     1940    @[redacted]@ 0x81fde308     2       70      0       False   2017-05-12 21:22:22.0000
00      N/A     Disabled
```

Interesting. This file name seems like it might be related to a famous ransomware from a few years ago. Let’s keep that in mind as we move through the investigation. While we’re at it, let’s also make a note of the *parent process ID* (PPID) too we’ll need it in **Question 12**.

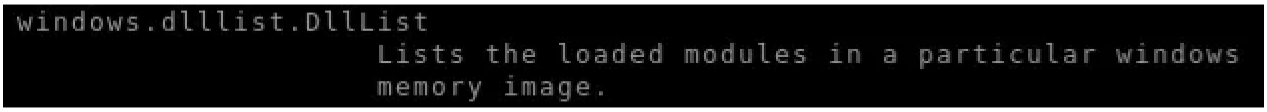


What suspicious process is running at PID 740 in Case 002?

[redacted]

## Question 10: What is the full path of the suspicious binary in PID 740 in Case 002?

Let's try to locate the file path of the suspicious binary. We'll first try to lean on our process plugs (**pslist, psscan, & pstree**) to see if we can find any information. Unfortunately, these commands aren't giving us much additional information so we will go back to the *Volatility 3* help and see if we can find a plugin that could help us.



```
windows.dlllist.DllList
                        Lists the loaded modules in a particular windows
                        memory image.
```

From the THM **Task 7** Module:

> *This plugin will list all DLLs associated with processes at the time of extraction. This can be especially useful once you have done further analysis and can filter output to a specific DLL that might be an indicator for a specific type of malware you believe to be present on the system.*

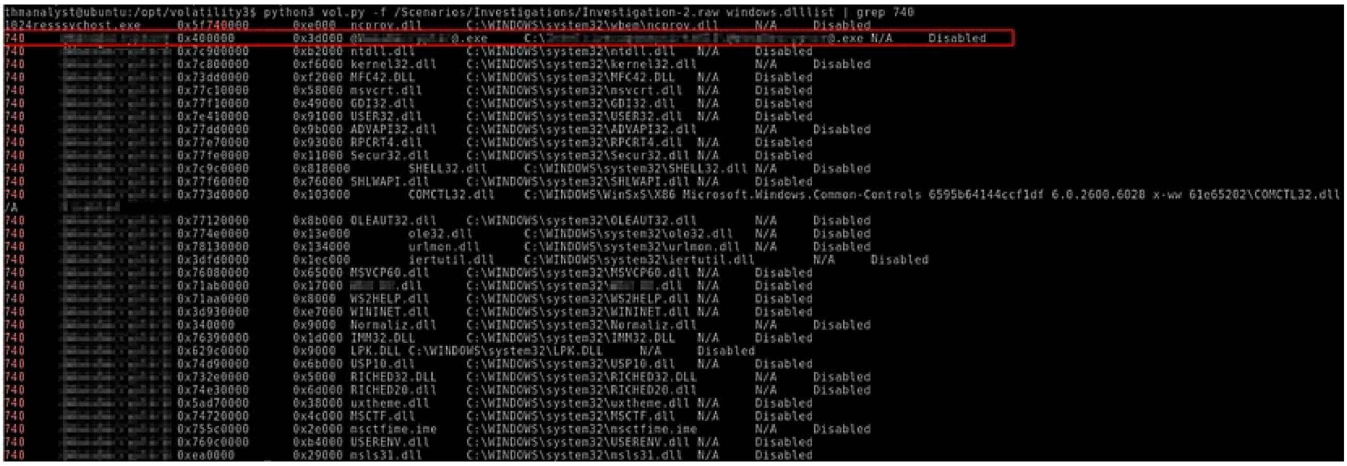
This could be useful to us from an investigative perspective but we also might get the file path of the binary that is loading the DLLs as well.

As a refresher, DLLs (Dynamic Link Library) are binary files that provide shared functionality for executables that can be called when required.

> *For the Windows operating systems, much of the functionality of the operating system is provided by DLL. Additionally, when you run a program on one of these Windows operating systems, much of the functionality of the program may be provided by DLLs. For example, some programs may contain many different modules, and each module of the program is contained and distributed in DLLs.*

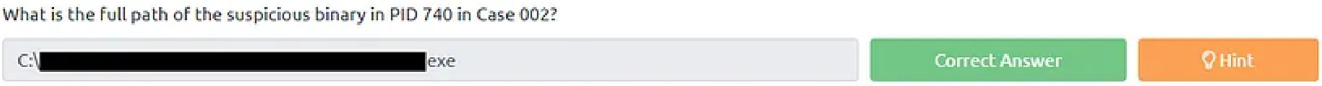
Let's give it a try and see what we can find.

```
python3 vol.py -f /Scenarios/Investigations/Investigation-2.raw windows.dlllist
```



```
thmanalyst@ubuntu:/opt/volatility3$ python3 vol.py -f /Scenarios/Investigations/Investigation-2.raw windows.dlllist | grep 740
1024resssvchost.exe     0x5f740000      0xe000  ncprov.dll      C:\WINDOWS\system32\wbem\ncprov.dll     N/A     Disabled
740     [redacted] 0x400000        0x3d000 [redacted].exe    C:\[redacted].exe N/A     Disabled
740     [redacted] 0x7c900000      0xb2000 ntdll.dll       C:\WINDOWS\system32\ntdll.dll   N/A     Disabled
740     [redacted] 0x7c800000      0xf6000 kernel32.dll    C:\WINDOWS\system32\kernel32.dll        N/A     Disabled
740     [redacted] 0x73dd0000      0xf2000 MFC42.DLL       C:\WINDOWS\system32\MFC42.DLL   N/A     Disabled
740     [redacted] 0x77c10000      0x58000 msvcrt.dll      C:\WINDOWS\system32\msvcrt.dll  N/A     Disabled
740     [redacted] 0x77f10000      0x49000 GDI32.dll       C:\WINDOWS\system32\GDI32.dll   N/A     Disabled
740     [redacted] 0x7e410000      0x91000 USER32.dll      C:\WINDOWS\system32\USER32.dll  N/A     Disabled
740     [redacted] 0x77dd0000      0x9b000 ADVAPI32.dll    C:\WINDOWS\system32\ADVAPI32.dll        N/A     Disabled
740     [redacted] 0x77e70000      0x93000 RPCRT4.dll      C:\WINDOWS\system32\RPCRT4.dll  N/A     Disabled
740     [redacted] 0x77fe0000      0x11000 Secur32.dll     C:\WINDOWS\system32\Secur32.dll N/A     Disabled
740     [redacted] 0x7c9c0000      0x818000        SHELL32.dll     C:\WINDOWS\system32\SHELL32.dll N/A     Disabled
740     [redacted] 0x77f60000      0x76000 SHLWAPI.dll     C:\WINDOWS\system32\SHLWAPI.dll N/A     Disabled
740     [redacted] 0x773d0000      0x103000        COMCTL32.dll    C:\WINDOWS\WinSxS\X86 Microsoft.Windows.Common-Controls 6595b64144ccf1df 6.0.2600.6028 x-ww 61e65202\COMCTL32.dll
/A      [redacted]
740     [redacted] 0x77120000      0x8b000 OLEAUT32.dll    C:\WINDOWS\system32\OLEAUT32.dll        N/A     Disabled
740     [redacted] 0x774e0000      0x13e000        ole32.dll       C:\WINDOWS\system32\ole32.dll   N/A     Disabled
740     [redacted] 0x78130000      0x134000        urlmon.dll      C:\WINDOWS\system32\urlmon.dll  N/A     Disabled
740     [redacted] 0x3dfd0000      0x1ec000        iertutil.dll    C:\WINDOWS\system32\iertutil.dll        N/A     Disabled
740     [redacted] 0x76080000      0x65000 MSVCP60.dll     C:\WINDOWS\system32\MSVCP60.dll N/A     Disabled
740     [redacted] 0x71ab0000      0x17000 [redacted].dll      C:\WINDOWS\system32\[redacted].dll  N/A     Disabled
740     [redacted] 0x71aa0000      0x8000  WS2HELP.dll     C:\WINDOWS\system32\WS2HELP.dll N/A     Disabled
740     [redacted] 0x3d930000      0xe7000 WININET.dll     C:\WINDOWS\system32\WININET.dll N/A     Disabled
740     [redacted] 0x340000        0x9000  Normaliz.dll    C:\WINDOWS\system32\Normaliz.dll        N/A     Disabled
740     [redacted] 0x76390000      0x1d000 IMM32.DLL       C:\WINDOWS\system32\IMM32.DLL   N/A     Disabled
740     [redacted] 0x629c0000      0x9000  LPK.DLL C:\WINDOWS\system32\LPK.DLL     N/A     Disabled
740     [redacted] 0x74d90000      0x6b000 USP10.dll       C:\WINDOWS\system32\USP10.dll   N/A     Disabled
740     [redacted] 0x732e0000      0x5000  RICHED32.DLL    C:\WINDOWS\system32\RICHED32.DLL        N/A     Disabled
740     [redacted] 0x74e30000      0x6d000 RICHED20.dll    C:\WINDOWS\system32\RICHED20.dll        N/A     Disabled
740     [redacted] 0x5ad70000      0x38000 uxtheme.dll     C:\WINDOWS\system32\uxtheme.dll N/A     Disabled
740     [redacted] 0x74720000      0x4c000 MSCTF.dll       C:\WINDOWS\system32\MSCTF.dll   N/A     Disabled
740     [redacted] 0x755c0000      0x2e000 msctfime.ime    C:\WINDOWS\system32\msctfime.ime        N/A     Disabled
740     [redacted] 0x769c0000      0xb4000 USERENV.dll     C:\WINDOWS\system32\USERENV.dll N/A     Disabled
740     [redacted] 0xea0000        0x29000 msls31.dll      C:\WINDOWS\system32\msls31.dll  N/A     Disabled
```

Awesome — this is exactly what we were looking for!



What is the full path of the suspicious binary in PID 740 in Case 002?

C:\[redacted]exe

## Questions 11 & 12:

**What is the parent process of PID 740 in Case 002?**

**What is the suspicious parent process PID connected to the decryptor in Case 002?**

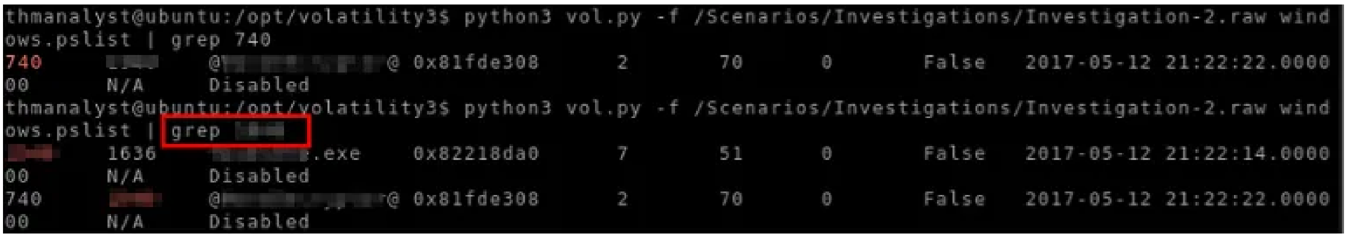
Alright, one step forward and two steps back. If you haven't cleared your terminal yet, lets scroll back up to your **pslist** output from **Question 9.**

Take a look at the *PPID* column for *PID 740*. Remember in **Question 9** where I mentioned we might want to make a note of the PPID of the suspicious process? That's what we need for **Question 12**.

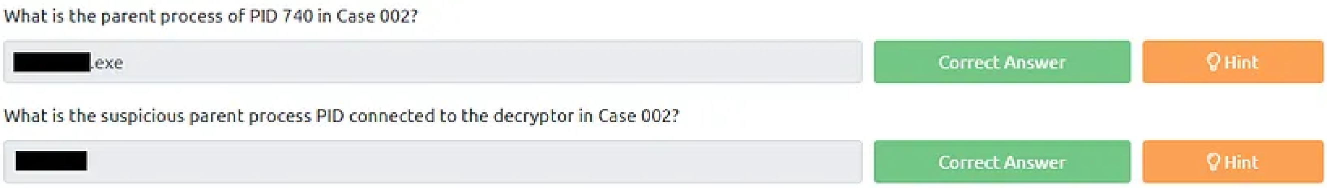
We will use **pslist** again and grep the parent process ID. After that, this becomes a simple matching game like we saw in Case 001.

```
python3 vol.py -f /Scenarios/Investigations/Investigation-2.raw windows.pslist | grep <ppid redacted>
```

When reviewing the output, **Question 11** is looking for the *ImageFileName* of the process. Have fun!



```
thmanalyst@ubuntu:/opt/volatility3$ python3 vol.py -f /Scenarios/Investigations/Investigation-2.raw wind
ows.pslist | grep 740
740     [redacted]    @[redacted]@ 0x81fde308      2        70      0        False   2017-05-12 21:22:22.0000
00      N/A      Disabled
thmanalyst@ubuntu:/opt/volatility3$ python3 vol.py -f /Scenarios/Investigations/Investigation-2.raw wind
ows.pslist | grep [redacted]
[redacted]    1636    [redacted].exe    0x82218da0      7        51      0        False   2017-05-12 21:22:14.0000
00      N/A      Disabled
740     [redacted]    @[redacted]@ 0x81fde308      2        70      0        False   2017-05-12 21:22:22.0000
00      N/A      Disabled
```



What is the parent process of PID 740 in Case 002?

[redacted].exe

What is the suspicious parent process PID connected to the decryptor in Case 002?

[redacted]

## Question 13: From our current information, what malware is present on the system in Case 002?

Let's get to *Google* for some research of the artifacts we've found so far. We'll start by searching for something broad, like the specific name of the executable that we discovered in **Question 9**.

We'll stumble across a few links, but I chose the threat report from **Mandiant** for this write-up.

Based on the report — we have already discovered some of these indicators of compromise (IOCs) on our victim system. I think that we have determined the malware strain that infected the victim system:

From our current information, what malware is present on the system in Case 002?

[redacted]

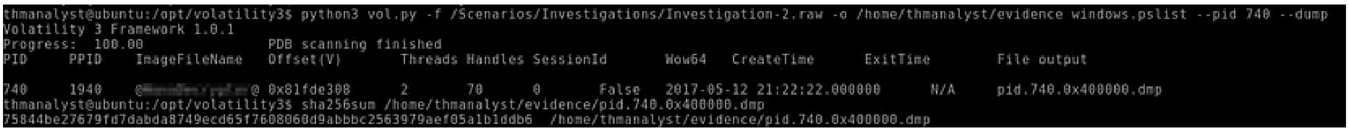
## Question 14: What DLL is loaded by the decryptor used for socket creation in Case 002?

Reading through the **Mandiant report** linked in **Question 13**, there are some mentions of socket functions but not necessarily what DLL is loaded specifically for socket creation. Let's do a little more manual work with *Volatility* and perform our own analysis.

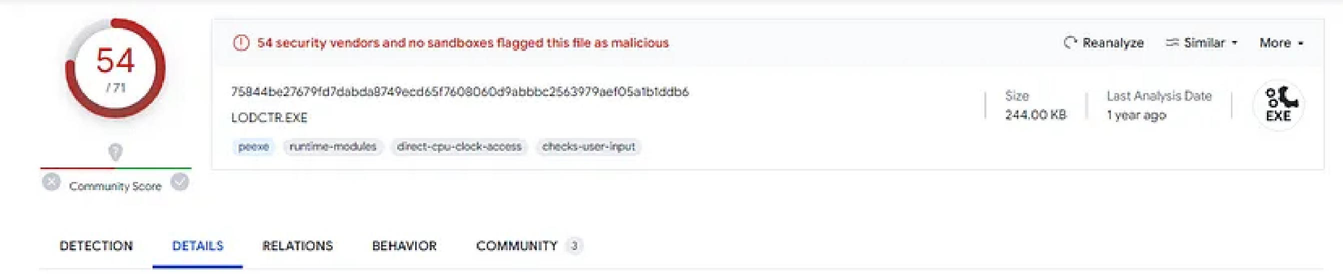
First, we will dump the process to see if I can learn anything on *VirusTotal* about any loaded DLLs by this executable. We're going to dump this to an output directory and then retrieve the file hash for comparison.

```
python3 vol.py -f /Scenarios/Investigations/Investigation-2.raw -o /home/thmana
```

```
sha256sum /home/thmanalyst/evidence/pid.740.0x400000.dmp
```



```
thmanalyst@ubuntu:/opt/volatility3$ python3 vol.py -f /Scenarios/Investigations/Investigation-2.raw -o /home/thmanalyst/evidence windows.pslist --pid 740 --dump
Volatility 3 Framework 1.0.1
Progress:  100.00               PDB scanning finished
PID     PPID    ImageFileName   Offset(V)       Threads Handles SessionId       Wow64   CreateTime      ExitTime        File output

740     1940    [redacted] 0x81fde308      2       70      0       False   2017-05-12 21:22:22.000000      N/A     pid.740.0x400000.dmp
thmanalyst@ubuntu:/opt/volatility3$ sha256sum /home/thmanalyst/evidence/pid.740.0x400000.dmp
75844be27679fd7dabda8749ecd65f7608060d9abbbc2563979aef05a1b1ddb6  /home/thmanalyst/evidence/pid.740.0x400000.dmp
```



54 security vendors and no sandboxes flagged this file as malicious

75844be27679fd7dabda8749ecd65f7608060d9abbbc2563979aef05a1b1ddb6

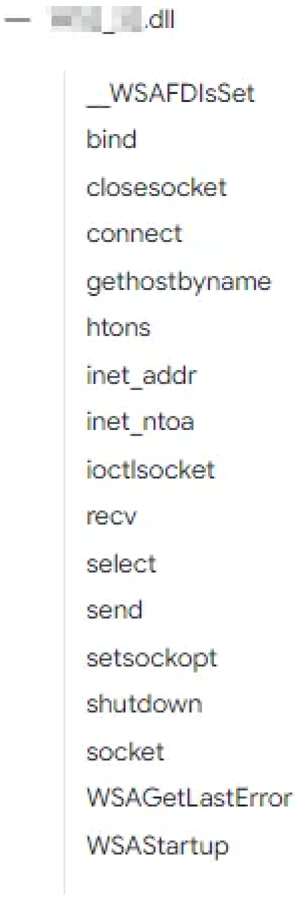
LODCTR.EXE

peexe runtime-modules direct-cpu-clock-access checks-user-input

Size 244.00 KB

Last Analysis Date 1 year ago

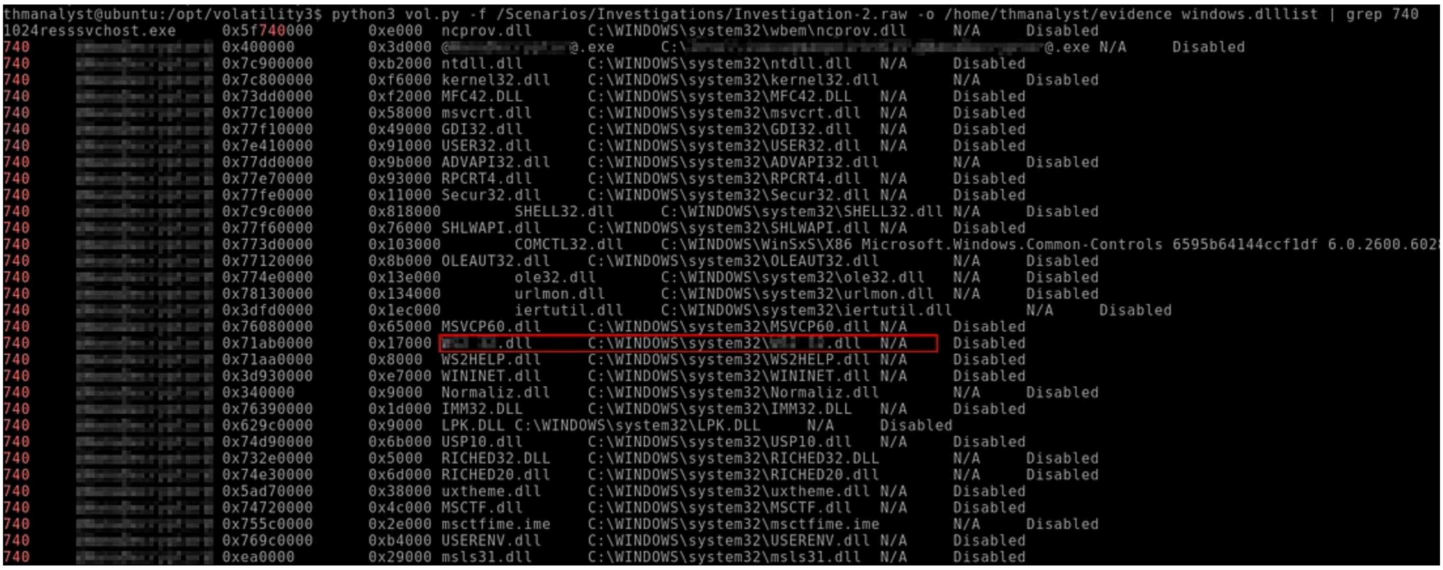
On the details tab, we'll scroll down to the imports and take a look at the list of DLLs. It might not be the most efficient way, but we can quickly expand on all of the imports and see if we can spot any network or socket functions specifically. Let's review the details; there is one that sticks out and looks like it could be relevant.



— [redacted]_[redacted].dll

```
__WSAFDIsSet
bind
closesocket
connect
gethostbyname
htons
inet_addr
inet_ntoa
ioctlsocket
recv
select
send
setsockopt
shutdown
socket
WSAGetLastError
WSAStartup
```

Now, let's return to the DLL list in our analysis environment and look at the output for this process again and see all of the DLLs loaded by this specific sample and verify we see the DLL here as well:

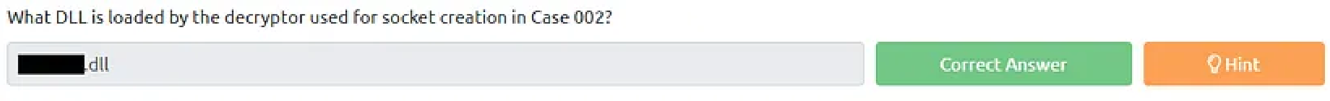


```
thmanalyst@ubuntu:/opt/volatility3$ python3 vol.py -f /Scenarios/Investigations/Investigation-2.raw -o /home/thmanalyst/evidence windows.dlllist | grep 740
1024resssvchost.exe     0x5f740000      0xe000  ncprov.dll      C:\WINDOWS\system32\wbem\ncprov.dll     N/A     Disabled
740     [redacted]  0x400000        0x3d000 @[redacted]@.exe     C:\[redacted]@.exe N/A     Disabled
740     [redacted]  0x7c900000      0xb2000 ntdll.dll       C:\WINDOWS\system32\ntdll.dll   N/A     Disabled
740     [redacted]  0x7c800000      0xf6000 kernel32.dll    C:\WINDOWS\system32\kernel32.dll        N/A     Disabled
740     [redacted]  0x73dd0000      0xf2000 MFC42.DLL       C:\WINDOWS\system32\MFC42.DLL   N/A     Disabled
740     [redacted]  0x77c10000      0x58000 msvcrt.dll      C:\WINDOWS\system32\msvcrt.dll  N/A     Disabled
740     [redacted]  0x77f10000      0x49000 GDI32.dll       C:\WINDOWS\system32\GDI32.dll   N/A     Disabled
740     [redacted]  0x7e410000      0x91000 USER32.dll      C:\WINDOWS\system32\USER32.dll  N/A     Disabled
740     [redacted]  0x77dd0000      0x9b000 ADVAPI32.dll    C:\WINDOWS\system32\ADVAPI32.dll        N/A     Disabled
740     [redacted]  0x77e70000      0x93000 RPCRT4.dll      C:\WINDOWS\system32\RPCRT4.dll  N/A     Disabled
740     [redacted]  0x77fe0000      0x11000 Secur32.dll     C:\WINDOWS\system32\Secur32.dll N/A     Disabled
740     [redacted]  0x7c9c0000      0x818000        SHELL32.dll     C:\WINDOWS\system32\SHELL32.dll N/A     Disabled
740     [redacted]  0x77f60000      0x76000 SHLWAPI.dll     C:\WINDOWS\system32\SHLWAPI.dll N/A     Disabled
740     [redacted]  0x773d0000      0x103000        COMCTL32.dll    C:\WINDOWS\WinSxS\X86 Microsoft.Windows.Common-Controls 6595b64144ccf1df 6.0.2600.602
740     [redacted]  0x77120000      0x8b000 OLEAUT32.dll    C:\WINDOWS\system32\OLEAUT32.dll        N/A     Disabled
740     [redacted]  0x774e0000      0x13e000        ole32.dll       C:\WINDOWS\system32\ole32.dll   N/A     Disabled
740     [redacted]  0x78130000      0x134000        urlmon.dll      C:\WINDOWS\system32\urlmon.dll  N/A     Disabled
740     [redacted]  0x3dfd0000      0x1ec000        iertutil.dll    C:\WINDOWS\system32\iertutil.dll        N/A     Disabled
740     [redacted]  0x76080000      0x65000 MSVCP60.dll     C:\WINDOWS\system32\MSVCP60.dll N/A     Disabled
740     [redacted]  0x71ab0000      0x17000 [redacted].dll       C:\WINDOWS\system32\[redacted].dll  N/A     Disabled
740     [redacted]  0x71aa0000      0x8000  WS2HELP.dll     C:\WINDOWS\system32\WS2HELP.dll N/A     Disabled
740     [redacted]  0x3d930000      0xe7000 WININET.dll     C:\WINDOWS\system32\WININET.dll N/A     Disabled
740     [redacted]  0x340000        0x9000  Normaliz.dll    C:\WINDOWS\system32\Normaliz.dll        N/A     Disabled
740     [redacted]  0x76390000      0x1d000 IMM32.DLL       C:\WINDOWS\system32\IMM32.DLL   N/A     Disabled
740     [redacted]  0x629c0000      0x9000  LPK.DLL C:\WINDOWS\system32\LPK.DLL     N/A     Disabled
740     [redacted]  0x74d90000      0x6b000 USP10.dll       C:\WINDOWS\system32\USP10.dll   N/A     Disabled
740     [redacted]  0x732e0000      0x5000  RICHED32.DLL    C:\WINDOWS\system32\RICHED32.DLL        N/A     Disabled
740     [redacted]  0x74e30000      0x6d000 RICHED20.dll    C:\WINDOWS\system32\RICHED20.dll        N/A     Disabled
740     [redacted]  0x5ad70000      0x38000 uxtheme.dll     C:\WINDOWS\system32\uxtheme.dll N/A     Disabled
740     [redacted]  0x74720000      0x4c000 MSCTF.dll       C:\WINDOWS\system32\MSCTF.dll   N/A     Disabled
740     [redacted]  0x755c0000      0x2e000 msctfime.ime    C:\WINDOWS\system32\msctfime.ime        N/A     Disabled
740     [redacted]  0x769c0000      0xb4000 USERENV.dll     C:\WINDOWS\system32\USERENV.dll N/A     Disabled
740     [redacted]  0xea0000        0x29000 msls31.dll      C:\WINDOWS\system32\msls31.dll  N/A     Disabled
```

Okay, I am thinking we may have found the answer but let's do some additional research. I'm going to try get a quick AI brief on this DLL from the *Microsoft Copilot* for *Edge* to before we validate the accuracy of the information through the reference links — it's always important to verify the accuracy of AI output.

> *The <REDACTED>.dll, also known as the Winsock2 DLL, is a dynamic link library file that provides essential functions for network communication in Windows operating systems*
>
> *<REDACTED>.dll plays a crucial role in managing network communication, ensuring compatibility, and facilitating efficient interactions between applications and service providers in the Windows environment*

Okay, confirmed! This seems like we can say with high confidence that the Winsock2 DLL is what is used for socket creation.



What DLL is loaded by the decryptor used for socket creation in Case 002?

[redacted].dll

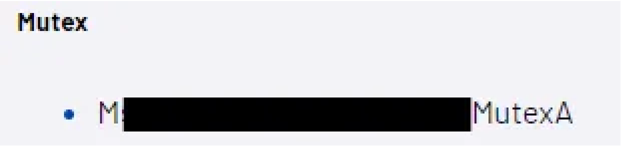
## Question 15: What mutex can be found that is a known indicator of the malware in question in Case 002?

This is an interesting question and is a new one for me! Let's do a quick *Google* refresher on a **mutex** for context.

Below is an excerpt from the SANS Blog:

> *Programs use mutex ("mutual exclusion") objects as a locking mechanism to serialize access to a resource on the system. Consider the following explanation by Microsoft: "For example, to prevent two threads from writing to shared memory at the same time, each thread waits for ownership of a mutex object before executing the code that accesses the memory. After writing to the shared memory, the thread releases the mutex object."*

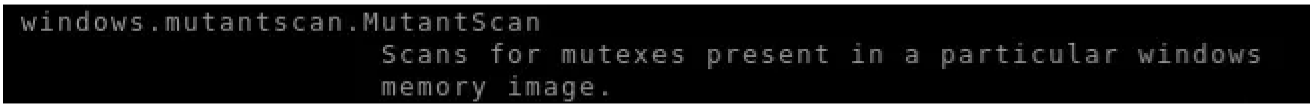
Now, let's check out the **Mandiant** report again and see if any of the heavy lifting has been done for us already. If we check out the file artifacts listed in the report, we see a *mutex* listed out.



**Mutex**

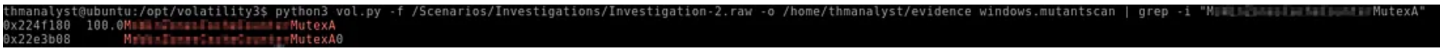
- M[redacted]MutexA

Okay, so in theory this is information that should be captured in the memory image and we should be able to find a mutex used by the malware. Let's check out the *Volatility* help and see if we can find any plugins that could help us validate this.



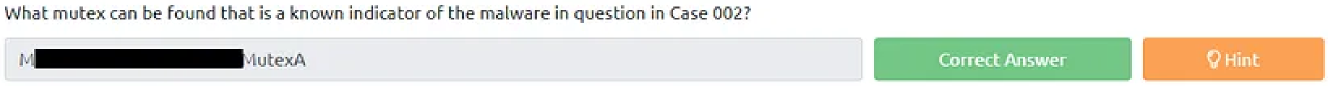
```
windows.mutantscan.MutantScan
                        Scans for mutexes present in a particular windows
                        memory image.
```

I’m sure there is a better way to utilize this plugin but in this case, let’s simply use the *Volatility 3* **windows.mutantscan** plugin to validate the presence of the *mutex* in our analysis sample against the report:



```
thmanalyst@ubuntu:/opt/volatility3$ python3 vol.py -f /Scenarios/Investigations/Investigation-2.raw -o /home/thmanalyst/evidence windows.mutantscan | grep -i "M[redacted]MutexA"
0x224f180  100.0M[redacted]MutexA
0x22e3b08        M[redacted]MutexA0
```

Great, we stumbled through this one! Let’s submit and confirm our suspicion.

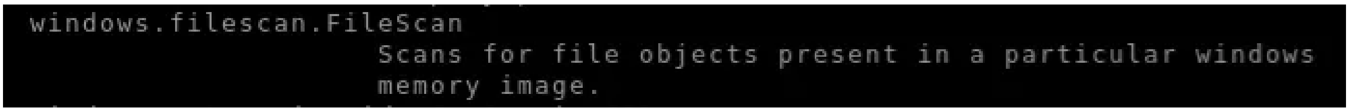


What mutex can be found that is a known indicator of the malware in question in Case 002?

M[redacted]MutexA

## Question 16: What plugin could be used to identify all files loaded from the malware working directory in Case 002?

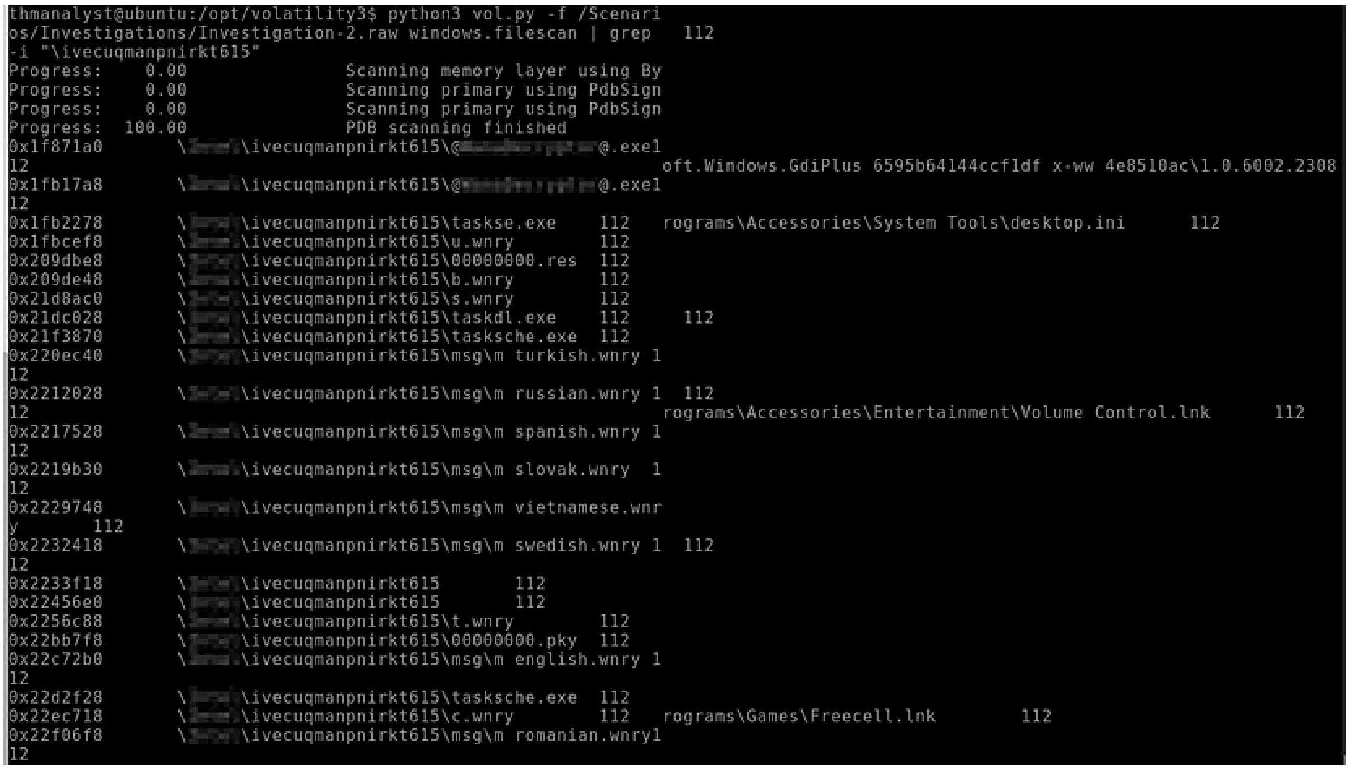
For the last question, we will return for the last time to our *Volatility 3* help file. Let’s see if there are any other plugins we can utilize for further analysis of the malware and search for the files loaded from the malware directory?



```
windows.filescan.FileScan
                        Scans for file objects present in a particular windows
                        memory image.
```

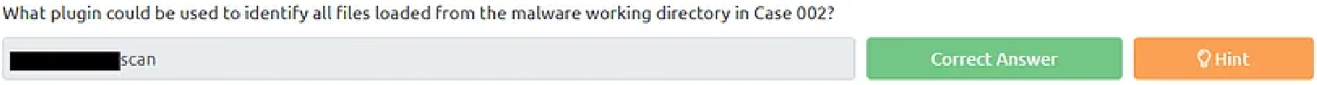
This plugin could be useful for further analysis especially if we run it against the malware directory that we found in **Question 10.** While not required for the challenge, let’s go ahead and run the command and grep the working directory:

```
python3 vol.py -f /Scenarios/Investigations/Investigation-2.raw windows.filesca
 "\ivecuqmanpnirkt615"
```



```
thmanalyst@ubuntu:/opt/volatility3$ python3 vol.py -f /Scenari
os/Investigations/Investigation-2.raw windows.filescan | grep   112
-i "\ivecuqmanpnirkt615"
Progress:    0.00               Scanning memory layer using By
Progress:    0.00               Scanning primary using PdbSign
Progress:    0.00               Scanning primary using PdbSign
Progress:  100.00               PDB scanning finished
0x1f871a0       \[redacted]\ivecuqmanpnirkt615\@[redacted]@.exe1
12                                                             oft.Windows.GdiPlus 6595b64144ccf1df x-ww 4e8510ac\1.0.6002.2308
0x1fb17a8       \[redacted]\ivecuqmanpnirkt615\@[redacted]@.exe1
12
0x1fb2278       \[redacted]\ivecuqmanpnirkt615\taskse.exe    112   rograms\Accessories\System Tools\desktop.ini      112
0x1fbcef8       \[redacted]\ivecuqmanpnirkt615\u.wnry        112
0x209dbe8       \[redacted]\ivecuqmanpnirkt615\00000000.res  112
0x209de48       \[redacted]\ivecuqmanpnirkt615\b.wnry        112
0x21d8ac0       \[redacted]\ivecuqmanpnirkt615\s.wnry        112
0x21dc028       \[redacted]\ivecuqmanpnirkt615\taskdl.exe    112     112
0x21f3870       \[redacted]\ivecuqmanpnirkt615\tasksche.exe  112
0x220ec40       \[redacted]\ivecuqmanpnirkt615\msg\m turkish.wnry 1
12
0x2212028       \[redacted]\ivecuqmanpnirkt615\msg\m russian.wnry 1  112
12                                                             rograms\Accessories\Entertainment\Volume Control.lnk      112
0x2217528       \[redacted]\ivecuqmanpnirkt615\msg\m spanish.wnry 1
12
0x2219b30       \[redacted]\ivecuqmanpnirkt615\msg\m slovak.wnry  1
12
0x2229748       \[redacted]\ivecuqmanpnirkt615\msg\m vietnamese.wnr
y        112
0x2232418       \[redacted]\ivecuqmanpnirkt615\msg\m swedish.wnry 1  112
12
0x2233f18       \[redacted]\ivecuqmanpnirkt615       112
0x22456e0       \[redacted]\ivecuqmanpnirkt615       112
0x2256c88       \[redacted]\ivecuqmanpnirkt615\t.wnry        112
0x22bb7f8       \[redacted]\ivecuqmanpnirkt615\00000000.pky  112
0x22c72b0       \[redacted]\ivecuqmanpnirkt615\msg\m english.wnry 1
12
0x22d2f28       \[redacted]\ivecuqmanpnirkt615\tasksche.exe  112
0x22ec718       \[redacted]\ivecuqmanpnirkt615\c.wnry        112   rograms\Games\Freecell.lnk        112
0x22f06f8       \[redacted]\ivecuqmanpnirkt615\msg\m romanian.wnry1
12
```

Wow! This gives us even more IOCs that we can use to validate our findings. For now, though — let's submit the answer to **Question 16** and wrap up these cases.



What plugin could be used to identify all files loaded from the malware working directory in Case 002?

[redacted]scan

## Conclusion:

There we have it — mission completed! Thank you to **TryHackMe** for the impressive room and challenge. This was a really great challenge to help me further explore *Volatility 3* and learn some new skills along the way and I hope that you learned something as well between the two cases. Personally, I especially appreciated the need to do external research and use some brain power on DFIR. Thank you for your time in checking out this (long) walkthrough and stumbling through the challenge with me. Stay curious!

Tryhackme Walkthrough Cybersecurity Blue Team Volatility

Follow

## Written by Drew Arpino

49 Followers · 2 Following

## No responses yet

What are your thoughts?

Respond

## More from Drew Arpino

Drew Arpino

## LetsDefend— Brute Force Attacks Challenge Walkthrough

Investigating a Brute Force Attack with Wireshark and Auth.log

Sep 16, 2024 5

Drew Arpino

## LetsDefend—MSHTML Challenge Walkthrough

Maldoc analysis using zipdump.py, re-search.py, & VirusTotal

Aug 26, 2024 3 1

Drew Arpino

## TryHackMe—Boogeyman 1 Challenge Walkthrough

Email, Endpoint, & Network Forensic Investigation using Thunderbird, LNKParse3, PowerShell Logs, JQ, & Wireshark

Aug 5, 2024 1

Drew Arpino

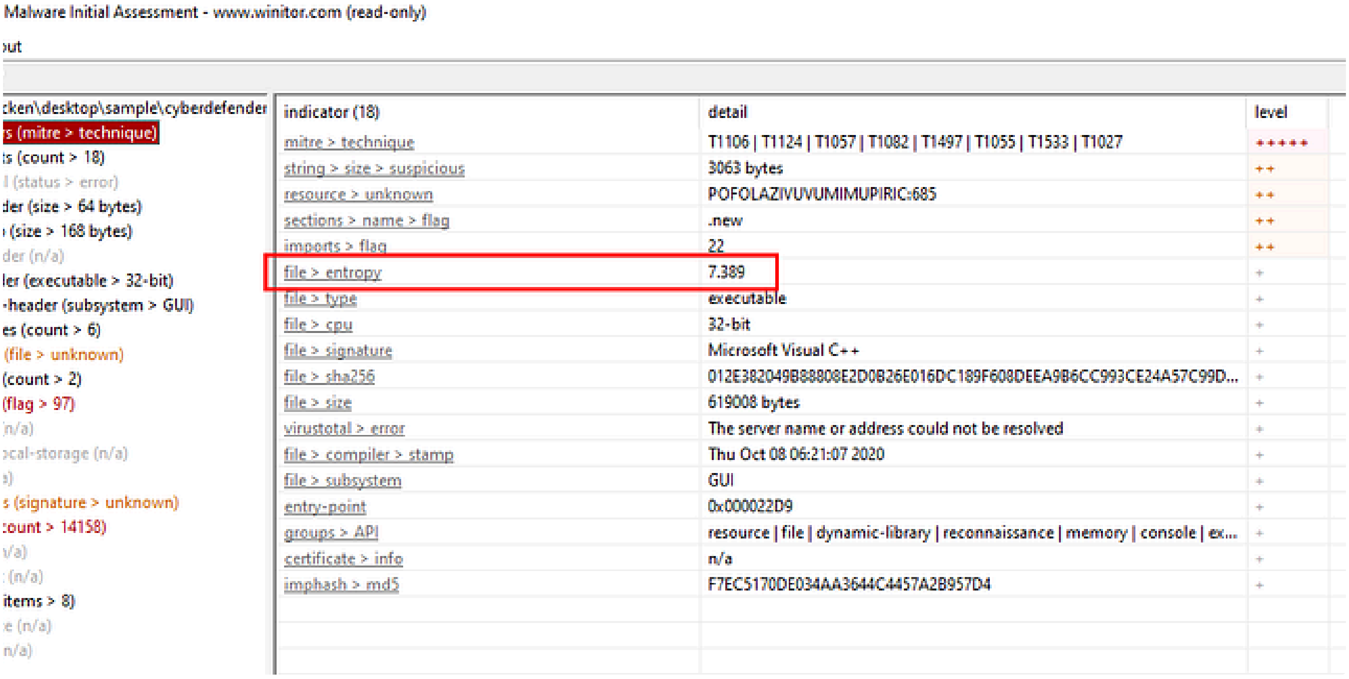
## TryHackMe—Basic Malware RE Walkthrough

Basic malware reverse engineering with Ghidra

Apr 22, 2024 1

See all from Drew Arpino

## Recommended from Medium



Malware Initial Assessment - www.winitor.com (read-only)

›ut

ːken\desktop\sample\cyberdefender
s (mitre > technique)
ːs (count > 18)
l (status > error)
der (size > 64 bytes)
› (size > 168 bytes)
der (n/a)
ler (executable > 32-bit)
-header (subsystem > GUI)
es (count > 6)
(file > unknown)
(count > 2)
(flag > 97)
n/a)
ocal-storage (n/a)
a)
s (signature > unknown)
count > 14158)
v/a)
: (n/a)
items > 8)
e (n/a)
n/a)

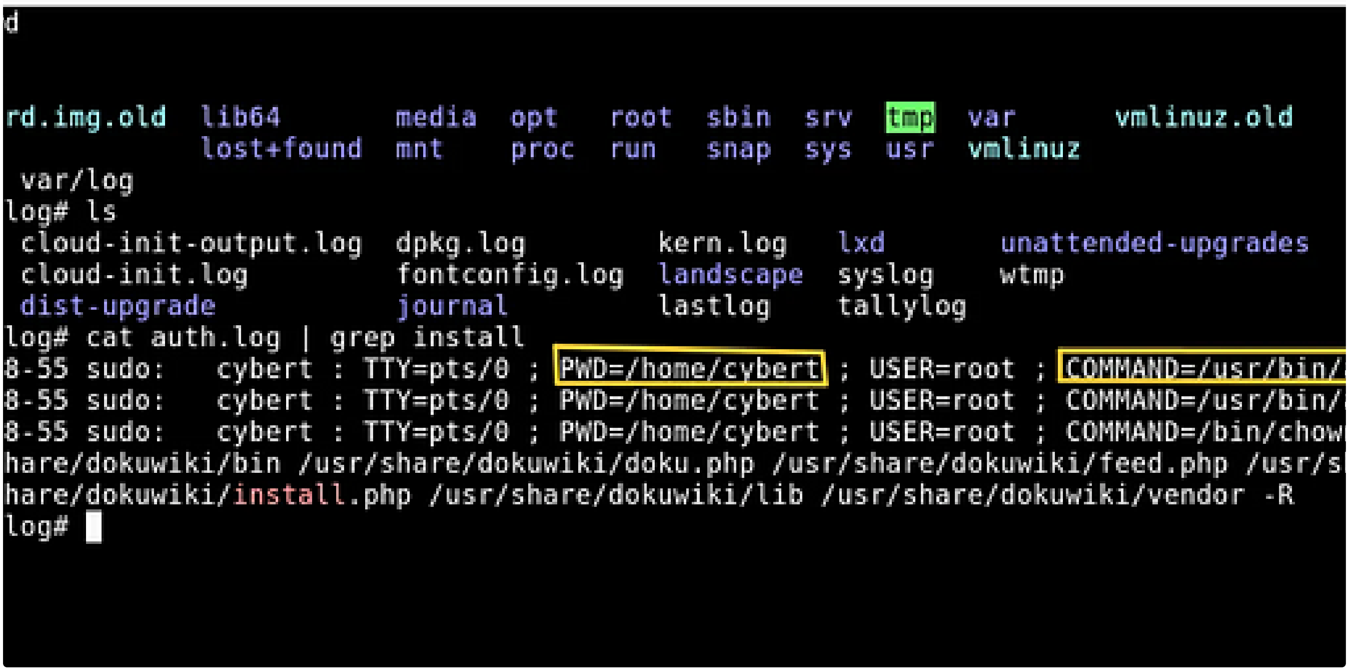
| indicator (18) | detail | level |
|---|---|---|
| mitre > technique | T1106 \| T1124 \| T1057 \| T1082 \| T1497 \| T1055 \| T1533 \| T1027 | +++++ |
| string > size > suspicious | 3063 bytes | ++ |
| resource > unknown | POFOLAZIVUVUMIMUPIRIC:685 | ++ |
| sections > name > flag | .new | ++ |
| imports > flag | 22 | ++ |
| file > entropy | 7.389 | + |
| file > type | executable | + |
| file > cpu | 32-bit | + |
| file > signature | Microsoft Visual C++ | + |
| file > sha256 | 012E382049B88808E2D0B26E016DC189F608DEEA9B6CC993CE24A57C99D... | + |
| file > size | 619008 bytes | + |
| virustotal > error | The server name or address could not be resolved | + |
| file > compiler > stamp | Thu Oct 08 06:21:07 2020 | + |
| file > subsystem | GUI | + |
| entry-point | 0x000022D9 | + |
| groups > API | resource \| file \| dynamic-library \| reconnaissance \| memory \| console \| ex... | + |
| certificate > info | n/a | + |
| imphash > md5 | F7EC5170DE034AA3644C4457A2B957D4 | + |

In System Weakness by Chicken0248

## [CyberDefenders Write-up] UnPackMe

Category: Malware Analysis

Oct 8, 2024 1



```
d

rd.img.old  lib64       media  opt   root  sbin  srv  tmp  var      vmlinuz.old
            lost+found  mnt    proc  run   snap  sys  usr  vmlinuz
 var/log
log# ls
 cloud-init-output.log  dpkg.log        kern.log  lxd       unattended-upgrades
 cloud-init.log         fontconfig.log  landscape  syslog   wtmp
 dist-upgrade           journal         lastlog   tallylog
log# cat auth.log | grep install
8-55 sudo:   cybert : TTY=pts/0 ; PWD=/home/cybert ; USER=root ; COMMAND=/usr/bin/
8-55 sudo:   cybert : TTY=pts/0 ; PWD=/home/cybert ; USER=root ; COMMAND=/usr/bin/
8-55 sudo:   cybert : TTY=pts/0 ; PWD=/home/cybert ; USER=root ; COMMAND=/bin/chow
hare/dokuwiki/bin /usr/share/dokuwiki/doku.php /usr/share/dokuwiki/feed.php /usr/s
hare/dokuwiki/install.php /usr/share/dokuwiki/lib /usr/share/dokuwiki/vendor -R
log# █
```

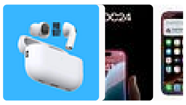
Dan Molina

## Disgruntled CTF Walkthrough

This is a great CTF on TryHackMe that can be accessed through this link here: https://tryhackme.com/room/disgruntled

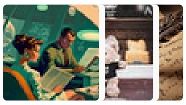
Oct 22, 2024

## Lists



### Tech & Tools

22 stories · 380 saves

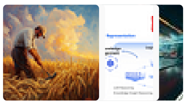


### Medium's Huge List of Publications Accepting Submissions

377 stories · 4345 saves

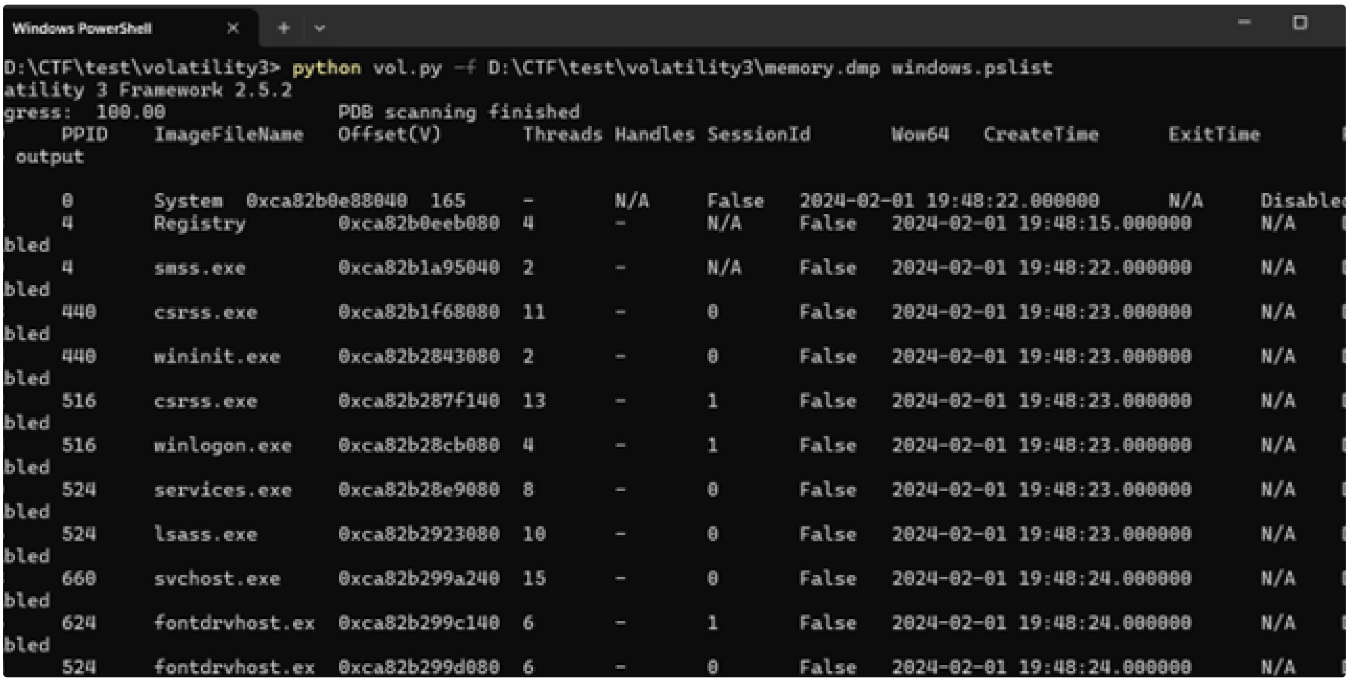
### Staff picks

796 stories · 1561 saves



### Natural Language Processing

1884 stories · 1529 saves



```
D:\CTF\test\volatility3> python vol.py -f D:\CTF\test\volatility3\memory.dmp windows.pslist
atility 3 Framework 2.5.2
gress:  100.00               PDB scanning finished
     PPID    ImageFileName   Offset(V)       Threads Handles SessionId       Wow64   CreateTime      ExitTime
 output

     0       System  0xca82b0e88040  165     -       N/A     False   2024-02-01 19:48:22.000000      N/A     Disable
     4       Registry        0xca82b0eeb080  4       -       N/A     False   2024-02-01 19:48:15.000000      N/A
bled
     4       smss.exe        0xca82b1a95040  2       -       N/A     False   2024-02-01 19:48:22.000000      N/A
bled
     440     csrss.exe       0xca82b1f68080  11      -       0       False   2024-02-01 19:48:23.000000      N/A
bled
     440     wininit.exe     0xca82b2843080  2       -       0       False   2024-02-01 19:48:23.000000      N/A
bled
     516     csrss.exe       0xca82b287f140  13      -       1       False   2024-02-01 19:48:23.000000      N/A
bled
     516     winlogon.exe    0xca82b28cb080  4       -       1       False   2024-02-01 19:48:23.000000      N/A
bled
     524     services.exe    0xca82b28e9080  8       -       0       False   2024-02-01 19:48:23.000000      N/A
bled
     524     lsass.exe       0xca82b2923080  10      -       0       False   2024-02-01 19:48:23.000000      N/A
bled
     660     svchost.exe     0xca82b299a240  15      -       0       False   2024-02-01 19:48:24.000000      N/A
bled
     624     fontdrvhost.ex  0xca82b299c140  6       -       1       False   2024-02-01 19:48:24.000000      N/A
bled
     524     fontdrvhost.ex  0xca82b299d080  6       -       0       False   2024-02-01 19:48:24.000000      N/A
```

Mo.Elshaheedy

## Ramnit Lab CyberDefenders

Category: Endpoint Forensics

Oct 10, 2024 52

Drew Arpino

## TryHackMe—Boogeyman 1 Challenge Walkthrough

Email, Endpoint, & Network Forensic Investigation using Thunderbird, LNKParse3, PowerShell Logs, JQ, & Wireshark

Aug 5, 2024 👏 1

IritT

## Windows Event Logs—Cyber Defense-Security Operations & Monitoring—TryHackMe Walkthrough

Introduction to Windows Event Logs and the tools to query them.

Oct 15, 2024

Zach Gillespie

### Conti: Ransomware Investigation with Splunk

In this Tryhackme challenge, an organization exchange server has been compromised with ransomware.

Aug 26, 2024 👏 1

See more recommendations