

Cyberdefender challenge DumpMe

1

The first question is linear, as soon as you download the file and unzip it with the password provided, run this command:

```
sha1sum Triage-Memory.mem
```

```
sansforensics@siftworkstation: ~  
$ sha1sum Triage-Memory.mem  
c95e8cc8c946f95a109ea8e47a6800de10a27abd  Triage-Memory.mem  
sansforensics@siftworkstation: ~
```

2

For the second question you can use imageinfo, and it'll output its suggestions, for a more accurate results use kdbgscan.

Imageinfo is usually enough because as soon as you try one profile, and the plugin doesn't work or gives some strange characters, you must need to try an other profile suggested by Imageinfo

```
vol.py -f Triage-Memory.mem imageinfo
```

```
sansforensics@siftworkstation: ~  
$ sha1sum Triage-Memory.mem  
c95e8cc8c946f95a109ea8e47a6800de10a27abd  Triage-Memory.mem  
sansforensics@siftworkstation: ~  
$ vol.py -f Triage-Memory.mem imageinfo  
Volatility Foundation Volatility Framework 2.6.1  
INFO : volatility.debug : Determining profile based on KDBG search...  
Suggested Profile(s) : Win7SP1x64, Win7SP0x64, Win2008R2SP0x64, Win2008R2SP1x64_24000, Win2008R2SP1x64_23418, Win2008R2SP1x64, Win7SP1x64_24000, Win7SP1x64_23418  
AS Layer1 : WindowsAMD64PagedMemory (Kernel AS)  
AS Layer2 : FileAddressSpace (/home/sansforensics/Triage-Memory.mem)  
PAE type : No PAE  
DTB : 0x187000L  
KDBG : 0xf800029f80a0L  
Number of Processors : 2  
Image Type (Service Pack) : 1  
KPCR for CPU 0 : 0xffffffff800029f9d00L
```

3 & 4

```
vol.py -f Triage-Memory.mem --profile=Win7SP1x64 pstree
```

0xfffffa8005419b30:chrome.exe	4240	3248	14	215	2019-03-22	05:35:17	UTC+0000
0xfffffa800540db30:chrome.exe	4520	3248	10	234	2019-03-22	05:35:18	UTC+0000
0xfffffa80052f0060:chrome.exe	2100	3248	2	59	2019-03-22	05:35:15	UTC+0000
0xfffffa80053cbb30:chrome.exe	4688	3248	13	168	2019-03-22	05:35:19	UTC+0000
0xfffffa800474c060:OUTLOOK.EXE	3688	1432	30	2023	2019-03-22	05:34:37	UTC+0000
0xfffffa8004798320:calc.exe	3548	1432	3	77	2019-03-22	05:34:43	UTC+0000
0xfffffa80053d3060:POWERPNT.EXE	4048	1432	23	765	2019-03-22	05:35:09	UTC+0000
0xfffffa8004905620:hfs.exe	3952	1432	6	214	2019-03-22	05:34:51	UTC+0000
0xfffffa8005a80060:wscript.exe	5116	3952	8	312	2019-03-22	05:35:32	UTC+0000
0xfffffa8005a1d9e0:UWkpfJfDzM.exe	3496	5116	5	109	2019-03-22	05:35:33	UTC+0000
0xfffffa8005bb0000:cmd.exe	4000	3496	1	33	2019-03-22	05:35:36	UTC+0000
0xfffffa80054f9060:notepad.exe	3032	1432	1	60	2019-03-22	05:32:22	UTC+0000
0xfffffa8005b49890:vmtoolsd.exe	1828	1432	6	144	2019-03-22	05:32:10	UTC+0000
0xfffffa800574fb30:taskmgr.exe	3792	1432	6	134	2019-03-22	05:34:38	UTC+0000
0xfffffa80053f83e0:EXCEL.EXE	1272	1432	21	789	2019-03-22	05:33:49	UTC+0000
0xfffffa8004083880:FTK Imager.exe	3192	1432	6	353	2019-03-22	05:35:12	UTC+0000
0xfffffa8003c72b30:System	4	0	87	547	2019-03-22	05:31:55	UTC+0000
0xfffffa8004616040:smss.exe	252	4	2	30	2019-03-22	05:31:55	UTC+0000
0xfffffa80050546b0:csrss.exe	332	324	10	516	2019-03-22	05:31:58	UTC+0000
0xfffffa8005259060:wininit.exe	380	324	3	78	2019-03-22	05:31:58	UTC+0000
0xfffffa8005680910:services.exe	476	380	12	224	2019-03-22	05:31:59	UTC+0000
0xfffffa8005409060:dllhost.exe	2072	476	13	194	2019-03-22	05:32:14	UTC+0000
0xfffffa8005b00060:wpnnetwk.exe	2628	476	9	210	2019-03-22	05:32:18	UTC+0000
0xfffffa800583db30:svchost.exe	1028	476	19	307	2019-03-22	05:32:05	UTC+0000
0xfffffa8005775b30:svchost.exe	796	476	15	368	2019-03-22	05:32:03	UTC+0000
0xfffffa8005050000:svchost.exe	1344	796	7	80	2019-03-22	05:32:07	UTC+0000

In This picture we can see multiple things:

- In the (1), the PID of notepad can be seen, that is 3032
- In the (2), the child process of wscript.exe, which is UWkpfJfDzM.exe, which is an unusual one probably malicious and abusing wscript.exe
- In the (3), the FTK Imager can be seen, which probably was the program used to capture the Ram on that machine to create the challenge

5

```
vol.py -f Triage-Memory.mem --profile=Win7SP1x64 netscan
```

```
rensics@slftworkstation: ~
py -f Triage-Memory.mem --profile=Win7SP1x64 netscan
lity Foundation Volatility Framework 2.6.1
(P)
Proto Local Address Foreign Address State Pid Owner Created
57300 UDPv4 10.0.0.101:55736 *: * 2888 svchost.exe 2019-03-22
5b4f0 UDPv6 ::1:55735 *: * 2888 svchost.exe 2019-03-22
5b790 UDPv6 fe80::7475:ef30:be18:7807:55734 *: * 2888 svchost.exe 2019-03-22
5d4b0 UDPv6 fe80::7475:ef30:be18:7807:1900 *: * 2888 svchost.exe 2019-03-22
5dec0 UDPv4 127.0.0.1:55737 *: * 2888 svchost.exe 2019-03-22
5e3f0 UDPv4 10.0.0.101:1900 *: * 2888 svchost.exe 2019-03-22
5eab0 UDPv6 ::1:1900 *: * 2888 svchost.exe 2019-03-22
64d70 UDPv4 127.0.0.1:1900 *: * 2888 svchost.exe 2019-03-22
2bcf0 TCPv4 -:49220 72.51.60.132:443 CLOSED 4048 POWERPNT.EXE
35790 TCPv4 -:49223 72.51.60.132:443 CLOSED 4048 POWERPNT.EXE
36470 TCPv4 -:49224 72.51.60.132:443 CLOSED 4048 POWERPNT.EXE
58010 UDPv4 127.0.0.1:55560 *: * 5116 wscript.exe 2019-03-22
05a50 UDPv4 0.0.0.0:5355 *: * 232 svchost.exe 2019-03-22
60be0 UDPv4 0.0.0.0:63790 *: * 504 2019-03-22
90ec0 UDPv4 0.0.0.0:5355 *: * 232 svchost.exe 2019-03-22
90ec0 UDPv6 ::1:5355 *: * 232 svchost.exe 2019-03-22
683e0 UDPv4 10.0.0.101:137 *: * 4 System 2019-03-22
94250 UDPv4 10.0.0.101:138 *: * 4 System 2019-03-22
97ec0 UDPv4 0.0.0.0:0 *: * 232 svchost.exe 2019-03-22
97ec0 UDPv6 ::1:0 *: * 232 svchost.exe 2019-03-22
1fb30 UDPv6 fe80::7475:ef30:be18:7807:546 *: * 764 svchost.exe 2019-03-22
18010 UDPv4 0.0.0.0:56372 *: * 1816 chrome.exe 2019-03-22
cd730 UDPv4 127.0.0.1:57374 *: * 1136 OfficeClickToR 2019-03-22
8e6a0 UDPv4 127.0.0.1:61704 *: * 3688 OUTLOOK.EXE 2019-03-22
d0bf0 UDPv4 127.0.0.1:55614 *: * 4048 POWERPNT.EXE 2019-03-22
```

6

```
vol.py -f Triage-Memory.mem --profile=Win7SP1x64 netscan
```

```

0x13e431820 TCPv4 0.0.0.0:49154 0.0.0.0:0 LISTENING 820 svchost.exe
0x13e57e010 TCPv4 10.0.0.101:139 0.0.0.0:0 LISTENING 4 System
0x13e71cef0 TCPv4 0.0.0.0:135 0.0.0.0:0 LISTENING 672 svchost.exe
0x13e720660 TCPv4 0.0.0.0:135 0.0.0.0:0 LISTENING 672 svchost.exe
0x13e720660 TCPv6 :::135 :::0 LISTENING 672 svchost.exe
0x13e72f010 TCPv4 0.0.0.0:49152 0.0.0.0:0 LISTENING 380 wininit.exe
0x13e72f6e0 TCPv4 0.0.0.0:49152 0.0.0.0:0 LISTENING 380 wininit.exe
0x13e72f6e0 TCPv6 :::49152 :::0 LISTENING 380 wininit.exe
0x13e770240 TCPv4 0.0.0.0:49153 0.0.0.0:0 LISTENING 764 svchost.exe
0x13e772980 TCPv4 0.0.0.0:49153 0.0.0.0:0 LISTENING 764 svchost.exe
0x13e772980 TCPv6 :::49153 :::0 LISTENING 764 svchost.exe
0x13ebb3010 TCPv4 0.0.0.0:49156 0.0.0.0:0 LISTENING 476 services.exe
0x13ebb3010 TCPv6 :::49156 :::0 LISTENING 476 services.exe
0x13ebcdef0 TCPv4 0.0.0.0:80 0.0.0.0:0 LISTENING 3952 hfs.exe
0x13e2348a0 TCPv4 -:49366 192.168.206.181:389 CLOSED 504
0x13e397190 TCPv4 10.0.0.101:49217 10.0.0.106:4444 ESTABLISHED 3496 WkpfJdZM.exe
0x13e3986d0 TCPv4 -:49378 213.209.1.129:25 CLOSED 504
0x13e3abae0 TCPv4 -:49226 72.51.60.132:443 CLOSED 4048 POWERPNT.EXE
0x13e3e7010 TCPv6 -:0 38db:7705:80fa:ffff:38db:7705:80fa:ffff:0 CLOSED 1136 OfficeClickToR
0x13e441830 TCPv6 -:0 382b:c703:80fa:ffff:382b:c703:80fa:ffff:0 CLOSED 1 ?RK???
0x13e4e4910 TCPv4 10.0.0.101:49208 52.109.12.6:443 CLOSED 504
0x13e55fae0 TCPv4 10.0.0.101:49209 52.96.44.162:443 CLOSED 504
0x13e71b540 TCPv4 -:0 104.208.112.5:0 CLOSED 1 ?RK???
0x13e73b560 TCPv4 -:49266 35.190.69.156:443 CLOSED 504
0x13e7c6010 TCPv4 10.0.0.101:49204 172.217.6.195:443 CLOSED 1816 chrome.exe
0x13ead7cf0 TCPv4 10.0.0.101:49202 172.217.10.68:443 CLOSED 1816 chrome.exe
0x13f5898a0 TCPv4 0.0.0.0:49156 0.0.0.0:0 LISTENING 476 services.exe
0x13f5899c0 TCPv4 0.0.0.0:445 0.0.0.0:0 LISTENING 4 System
0x13f5899c0 TCPv6 :::445 :::0 LISTENING 4 System
0x13f4facf0 TCPv4 10.0.0.101:49262 52.109.12.6:443 ESTABLISHED 3688 OUTLOOK.EXE
0x13f50a010 TCPv4 -:49265 213.186.33.3:443 CLOSED 504
0x13f5289f0 TCPv4 -:49234 72.51.60.133:80 CLOSED 3688 OUTLOOK.EXE
0x13f7b4ec0 UDPv4 0.0.0.0:55707 *: * 232 svchost.exe 2019-03-22 05:45:44 UTC+0000
0x13f7e8670 UDPv4 127.0.0.1:59411 *: * 3576 texplore.exe 2019-03-22 05:34:49 UTC+0000

```

It's possible to see that that strange process, has an **ESTABLISHED** connection to a remote ip on port **4444**, this port is an infamous port since it's what a lazy attacker will put on a metasploit meterpreter attack, deliver that payload and he'll put that port on listening with 'nc -lvnp 4444' to get a reverse shell/exfiltrate some data

7

The method to obtain which dll are run and by which process is linear, using the plugin dlllist will provide that information along with the cmdline for that process

```
vol.py -f Triage-Memory.mem --profile=Win7SP1x64 dlllist
```

```

690ea20bc3bdfb328e23005d9a80c290 executable.3496.exe
samsforensics@siftworkstation: ~
$ vol.py -f Triage-Memory.mem --profile=Win7SP1x64 dlllist
Volatility Foundation Volatility Framework 2.6.1
*****
System pid: 4
Unable to read PEB for task.
*****
smss.exe pid: 252
Command line : \SystemRoot\System32\smss.exe

Base                               Size          LoadCount LoadTime                               Path
-----
0x0000000048430000                  0x20000          0xffff 1970-01-01 00:00:00 UTC+0000    \SystemRoot\System32\smss.exe
0x0000000077260000                  0x1a9000          0xffff 1970-01-01 00:00:00 UTC+0000    C:\Windows\SYSTEM32\ntdll.dll
*****
csrss.exe pid: 332
Command line : %SystemRoot%\system32\csrss.exe ObjectDirectory=\Windows SharedSection=1024,20480,768 Windows=On SubSystemType=Windows
DllInitialization,3 ServerDll=winsrv:ConServerDllInitialization,2 ServerDll=sxssrv,4 ProfileControl=Off MaxRequestThreads=16
Service Pack 1

Base                               Size          LoadCount LoadTime                               Path
-----
0x0000000049dc0000                  0x6000          0xffff 1970-01-01 00:00:00 UTC+0000    C:\Windows\system32\csrss.exe
0x0000000077260000                  0x1a9000          0xffff 1970-01-01 00:00:00 UTC+0000    C:\Windows\SYSTEM32\ntdll.dll
0x000007fefdc230000                  0x13000          0xffff 2019-03-22 05:31:58 UTC+0000    C:\Windows\system32\CSRSSRV.DLL
0x000007fefdc210000                  0x11000          0x4 2019-03-22 05:31:58 UTC+0000    C:\Windows\system32\basesrv.DLL
0x000007fefdc1d0000                  0x38000          0x2 2019-03-22 05:31:58 UTC+0000    C:\Windows\system32\winsrv.DLL
0x0000000077160000                  0xfa000          0xb 2019-03-22 05:31:58 UTC+0000    C:\Windows\system32\USER32.dll
0x000007fefef60000                  0x67000          0xc 2019-03-22 05:31:58 UTC+0000    C:\Windows\system32\GDI32.dll
0x0000000077040000                  0x11f000          0x63 2019-03-22 05:31:58 UTC+0000    C:\Windows\SYSTEM32\kernel32.dll
0x000007fefdc380000                  0x6c000          0x132 2019-03-22 05:31:58 UTC+0000    C:\Windows\system32\KERNELBASE.dll
0x000007feff150000                  0xe000          0x3 2019-03-22 05:31:58 UTC+0000    C:\Windows\system32\LPK.dll
0x000007fefe5e0000                  0xc9000          0x3 2019-03-22 05:31:58 UTC+0000    C:\Windows\system32\USP10.dll
0x000007fefdc6b0000                  0x9f000          0x5 2019-03-22 05:31:58 UTC+0000    C:\Windows\system32\msvcrt.dll
0x000007fefdc1c0000                  0xc000          0x1 2019-03-22 05:31:58 UTC+0000    C:\Windows\system32\sxssrv.DLL
0x000007fefdc6b0000                  0x91000          0x1 2019-03-22 05:31:59 UTC+0000    C:\Windows\system32\sxs.dll
0x000007fefe7a0000                  0x12d000          0x3 2019-03-22 05:31:59 UTC+0000    C:\Windows\system32\RPCRT4.dll
0x000007fefdc0a0000                  0xf000          0x2 2019-03-22 05:31:59 UTC+0000    C:\Windows\system32\CRYPTBASE.dll
0x000007fefe970000                  0xdb000          0x1 2019-03-22 05:32:13 UTC+0000    C:\Windows\system32\ADVAPI32.dll
0x000007feff160000                  0x1f000          0x4 2019-03-22 05:32:13 UTC+0000    C:\Windows\SYSTEM32\sechost.dll
*****

```

A grep in the output will be enough to count how many process have loaded that dll

```
vol.py -f Triage-Memory.mem --profile=Win7SP1x64 dlllist | grep VCRUNTIME140
```

```
sansforensics@siftworkstation: ~
$ vol.py -f Triage-Memory.mem --profile=Win7SP1x64 dlllist | grep VCRUNTIME140
Volatility Foundation Volatility Framework 2.6.1
0x000007fefa5c0000 0x16000 0xffff 2019-03-22 05:32:05 UTC+0000 C:\Program Files\Common Files\Microsoft Shared\ClickToRun\VCRUNTIME140.dll
0x00000000745f0000 0x15000 0xffff 2019-03-22 05:33:49 UTC+0000 C:\Program Files (x86)\Microsoft Office\root\Office16\VCRUNTIME140.dll
0x00000000745f0000 0x15000 0xffff 2019-03-22 05:34:37 UTC+0000 C:\Program Files (x86)\Microsoft Office\root\Office16\VCRUNTIME140.dll
0x00000000745f0000 0x15000 0x3 2019-03-22 05:34:49 UTC+0000 C:\Program Files (x86)\Microsoft Office\root\Office16\VCRUNTIME140.dll
0x00000000745f0000 0x15000 0xffff 2019-03-22 05:35:09 UTC+0000 C:\Program Files (x86)\Microsoft Office\root\Office16\VCRUNTIME140.dll
sansforensics@siftworkstation: ~
```

8

For this flag, we must dump the process, using the following command will exactly do that:

```
vol.py -f Triage-Memory.mem --profile=Win7SP1x64 procdump --pid 3496 --dump-dir .
```

As soon as that process finishes, it'll tell you how the file was named and you can md5sum it

```
sansforensics@siftworkstation: ~
$ vol.py -f Triage-Memory.mem --profile=Win7SP1x64 procdump --pid 3496 --dump-dir .
Volatility Foundation Volatility Framework 2.6.1
Process(V)          ImageBase          Name                Result
-----
0xfffffa8005a1d9e0 0x0000000000400000 UWkpjFjDzM.exe      OK: executable.3496.exe
sansforensics@siftworkstation: ~
$ md5sum executable.3496.exe
690ea20bc3bdfb328e23005d9a80c290 executable.3496.exe
sansforensics@siftworkstation: ~
```

Flag: 690ea20bc3bdfb328e23005d9a80c290

9

This is asking for the hashes of Bob's password, having access to that system, you could mount it and then using this command

```
sudo grep 'Bob' /etc/shadow | cut -d ':' -f 2
```

Having the memory and already using volatility, a simple hashdump was enough to get the flag

```
sansforensics@siftworkstation: ~
$ vol.py -f Triage-Memory.mem --profile=Win7SP1x64 hashdump
Volatility Foundation Volatility Framework 2.6.1
Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
Bob:1000:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
sansforensics@siftworkstation: ~
$
```

Flag: aad3b435b51404eeaad3b435b51404ee

10 & 11

Realizing this and the next question was about VAD informations, it was easier to run the same command once and the grepping onto that two times

```
sansforensics@siftworkstation: ~
$ vol.py -f Triage-Memory.mem --profile=Win7SP1x64 vadinfo > vadfiles.txt
Volatility Foundation Volatility Framework 2.6.1
```

For the first a simple grep on that file is enough (I've used the -C flag to output the 20 lines before and after the match, because I didn't remember exactly where the information resided)

```
cat vadfiles.txt | grep -C 20 0xfffffa800577ba10
```

```
VAD node @ 0xfffffa800577ba10 Start 0x0000000000003000 End 0x00000000000033fff Tag Vad
Flags: NoChange: 1, Protection: 1
Protection: PAGE_READONLY
Vad Type: VadNone
ControlArea @fffffa8005687a50 Segment fffff8a000c4f870
NumberOfSectionReferences: 1 NumberOfPfnReferences: 0
NumberOfMappedViews: 29 NumberOfUserReferences: 30
Control Flags: Commit: 1
```

For the second one, since the information requested operates on two parameters, Start and End, it is needed multiple grep commands, therefore the command was:

```
cat vadfiles.txt | grep -C 20 0x000000000033c000 | grep -C 20 0x000000000033dffff
```

```
VAD node @ 0xfffffa80052652b0 Start 0x000000000033c000 End 0x000000000033dffff Tag VadS
Flags: CommitCharge: 32, PrivateMemory: 1, Protection: 24
Protection: PAGE_NOACCESS
Vad Type: VadNone
```

12

The cmdline action would give the answer away:

```
vol.py -f Triage-Memory.mem --profile=Win7SP1x64 cmdline
```

There could be another way, but since the question was "There was a VBS script that ran on the machine. What is the name of the script? (submit without file extension)".

I find it likely to be about the malicious files, but there;s no reference to it so using all the output of 'cmdline' should be necessary

However since the pids of the malicous processes are already known, we could use cmdline's option '--pid'.

- hfs.exe (wscript.exe's parent process)
- wscript.exe (vhjReUDEuumrX.exe's parent process)
- vhjReUDEuumrX.exe (The malious file itself)

```
vol.py -f Triage-Memory.mem --profile=Win7SP1x64 cmdline --pid 5116,3952,3496
```

```
sansforensics@siftworkstation: ~
$ vol.py -f Triage-Memory.mem --profile=Win7SP1x64 cmdline --pid 5116,3952,3496
Volatility Foundation Volatility Framework 2.6.1
*****
hfs.exe pid: 3952
Command line : "C:\Users\Bob\Desktop\hfs.exe"
*****
wscript.exe pid: 5116
Command line : "C:\Windows\System32\wscript.exe" //B //NOLOGO %TEMP%\vhjReUDEuumrX.vbs
*****
JWkpjFjDzM.exe pid: 3496
Command line : "C:\Users\Bob\AppData\Local\Temp\rad93398.tmp\UWkpjFjDzM.exe"
sansforensics@siftworkstation: ~
```

If this had no answer, removing the '--pid' option and running cmdline again will get us the cmdline on every process.


```

*****
chrome.exe pid: 4232
Command line : "C:\Program Files (x86)\Google\Chrome\Application\chrome.exe" --type=renderer --field-trial-handle=924,2132560186875629139,181
token=16174022435611898554 --lang=en-US --enable-offline-auto-reload --enable-offline-auto-reload-visible-only --device-scale-factor=1 --num-
el-token=16174022435611898554 --renderer-client-id=8 --no-v8-untrusted-code-mitigations --mojo-platform-channel-handle=2528 /prefetch:1
*****
chrome.exe pid: 4240
Command line : "C:\Program Files (x86)\Google\Chrome\Application\chrome.exe" --type=renderer --field-trial-handle=924,2132560186875629139,181
token=6187966837059053897 --lang=en-US --extension-process --enable-offline-auto-reload --enable-offline-auto-reload-visible-only --device-sc
rvice-request-channel-token=6187966837059053897 --renderer-client-id=4 --no-v8-untrusted-code-mitigations --mojo-platform-channel-handle=2536
*****
chrome.exe pid: 4520
Command line :
*****
chrome.exe pid: 4688
Command line : "C:\Program Files (x86)\Google\Chrome\Application\chrome.exe" --type=renderer --field-trial-handle=924,2132560186875629139,181
ompositing --service-pipe-token=2094820586012107996 --lang=en-US --enable-offline-auto-reload --enable-offline-auto-reload-visible-only --dev
1 --service-request-channel-token=2094820586012107996 --renderer-client-id=13 --no-v8-untrusted-code-mitigations --mojo-platform-channel-hand
*****
wscript.exe pid: 5116
Command line : "C:\Windows\System32\wscript.exe" //B //NOLOGO %TEMP%\vhjReUDEuumrX.vbs
*****
UwkpjFjDzM.exe pid: 3496
Command line : "C:\Users\Bob\AppData\Local\Temp\rad93398.tmp\UwkpjFjDzM.exe"
*****
cmd.exe pid: 4660
Command line : C:\Windows\system32\cmd.exe
*****
conhost.exe pid: 4656
Command line : [?]\C:\Windows\system32\conhost.exe "-2105419398-1914772985-1870609020-898984210-362940684623073995-430731961-49370805
sansforensics@siftworkstation: ~
$ vol.py -f Triage-Memory.mem --profile=Win7SP1x64 cmdline

```

13

The command to find the flag was taking so much time that I started writing this notes, the process was still alive so I looked if it was writing anything

So at first I checked if it did write something and if the process was alive, stopped it and then relaunched

```

sansforensics@siftworkstation: ~
$ ^C
sansforensics@siftworkstation: ~
$ ^C
sansforensics@siftworkstation: ~
$ vol.py -f Triage-Memory.mem --profile=Win7SP1x64 timeliner > timeline
bash: timeline: cannot overwrite existing file
sansforensics@siftworkstation: ~
$ vol.py -f Triage-Memory.mem --profile=Win7SP1x64 timeliner > timeline2
ShowApplications ation Volatility Framework 2.6.1

```

it was still taking so much time so I've started looking at what was writing, it seemed stucked at Bob's ntuser.dat but luckily for us, the flag was beyond that time so it must already be on that file

```

sansforensics@siftworkstation: ~
$ tail -f timeline2
2019-03-22 05:42:25 UTC+0000|[ _HBASE_BLOCK TimeStamp]| [?]\C:\Windows\ServicePro
files\LocalService\NTUSER.DAT|
2019-03-22 04:23:30 UTC+0000|[ _CMHIVE LastWriteTime]| [?]\C:\Windows\ServicePro
files\LocalService\NTUSER.DAT|
2019-03-22 05:43:11 UTC+0000|[ _HBASE_BLOCK TimeStamp]| [?]\C:\Users\Bob\AppData\
Local\Microsoft\Windows\UserClass.dat|
2019-03-22 04:23:26 UTC+0000|[ _CMHIVE LastWriteTime]| [?]\C:\Users\Bob\AppData\L
ocal\Microsoft\Windows\UserClass.dat|
1970-01-01 00:00:00 UTC+0000|[ _HBASE_BLOCK TimeStamp]| [no name]|
1970-01-01 00:00:00 UTC+0000|[ _CMHIVE LastWriteTime]| [no name]|
2019-03-22 05:44:16 UTC+0000|[ _HBASE_BLOCK TimeStamp]| \SystemRoot\System32\Conf
ig\SOFTWARE|
2019-03-22 04:23:30 UTC+0000|[ _CMHIVE LastWriteTime]| \SystemRoot\System32\Conf
ig\SOFTWARE|
2019-03-22 05:46:12 UTC+0000|[ _HBASE_BLOCK TimeStamp]| [?]\C:\Users\Bob\ntuser.d
at|
2019-03-22 04:23:26 UTC+0000|[ _CMHIVE LastWriteTime]| [?]\C:\Users\Bob\ntuser.da
t|

```

A grep on patter given, which was the time would give away the flag

```

sansforensics@siftworkstation: ~
$ grep "2019-03-07 23:06:58" timeline2
2019-03-07 23:06:58 UTC+0000|[SHIMCACHE]| [?]\C:\Program Files (x86)\Microsoft\S
kype for Desktop\Skype.exe|
sansforensics@siftworkstation: ~
$

```

14

```
vol.py -f Triage-Memory.mem --profile=Win7SP1x64 procdump --pid 3032 --dump-dir .
```

```
sansforensics@siftworkstation: ~
$ vol.py -f Triage-Memory.mem --profile=Win7SP1x64 memdump --pid 3032 --dump-dir .
Volatility Foundation Volatility Framework 2.6.1
*****
Writing notepad.exe [ 3032] to 3032.dmp
sansforensics@siftworkstation: ~
$
```

```
strings -e l 3032.dmp | grep -i flag
```

will give away so many result that it'll be mostly just noise. The flag is first and a simple `head` would suffice but I did not know that, instead I enhanced the grep based on what cyberdefenders format would expect:

Skype.exe

#14 What was written in notepad.exe at the

Format: flag<*****_**_*****>

Therefore I grepped as the following command and

```
strings -e l 3032.dmp | grep -i 'flag<'
```

and reduced the results from 374 to 5

```
[Roaming::CNoThrowMruItem::GetOpenFlags]
Software\Microsoft\Windows NT\CurrentVersion\AppCompat
outlook.flags
~-1-5-21-1497316740-357279761-3945674337-1000\Software
SppNotificationFlags
flag<TheK>
flag<TheK>
flag
flag@
sansforensics@siftworkstation: ~
$ strings -e l 3032.dmp | grep -i 'flag<'
flag<REDBULL_IS_LIFE>
flag<Th>
flag<Th>
flag<TheK>
flag<TheK>
```

15

```
vol.py -f Triage-Memory.mem mftparser > mft.txt && grep -C 20 '59045' mft.txt | grep -iC 20
'record number'
```

\$DATA

\$OBJECT_ID

Object ID: 40000000-0000-0000-0010-000000000000
Birth Volume ID: 19050000-0000-0000-1905-000000000000
Birth Object ID: 31015ed0-1900-ffff-ffff-ffff82794711
Birth Domain ID: ffffffff-8279-4711-0000-000000000000

MFT entry found at offset 0x2193d400
Attributes: In Use & File
Record Number: 59045
Link count: 2

\$STANDARD_INFORMATION

Creation	Modified	MFT Altered	Access Date	Type
2019-03-17 06:50:07 UTC+0000	2019-03-17 07:04:43 UTC+0000	2019-03-17 07:04:43 UTC+0000	2019-03-17 07:04:42 UTC+0000	Archive

\$FILE_NAME

Creation	Modified	MFT Altered	Access Date	Name/Path
2019-03-17 06:50:07 UTC+0000	2019-03-17 07:04:43 UTC+0000	2019-03-17 07:04:43 UTC+0000	2019-03-17 07:04:42 UTC+0000	Users\Bob\DOCUME~1\EMPLOY~1\EMPLOY~1.XLS

\$FILE_NAME

Creation	Modified	MFT Altered	Access Date	Name/Path
2019-03-17 06:50:07 UTC+0000	2019-03-17 07:04:43 UTC+0000	2019-03-17 07:04:43 UTC+0000	2019-03-17 07:04:42 UTC+0000	Users\Bob\DOCUME~1\EMPLOY~1\EmployeeInformation.xlsx

\$OBJECT_ID

Object ID: 00fe50d2-4841-e911-8751-000c2958bc5f

I think this one as the bonus question, because it's already kinda known the malicious PID, 3496 the UWkpjFjDzM.exe process. The "4444", as previously mentioned is a dead giveaway of a metasploit payload, is the default listener port for its meterpreter staging and payloads.