CyberDefenders: DumpMe Lab

The following writeup is for DumpMe Lab on CyberDefenders, it involves investigating a memory dump using Volatility 2. Those who enjoy memory forensics should definitely give this a go, it covers a range of skills and only takes around an hour or so.

Scenario: A SOC analyst took a memory dump from a machine infected with a meterpreter malware. As a Digital Forensicators, your job is to analyze the dump, extract the available indicators of compromise (IOCs) and answer the provided questions.

What is the SHA1 hash of Triage-Memory.mem (memory dump)?

To generate the SHA1 hash of the memory dump, we can use the sha1sum utility:

```
remnux@remnux:~/Desktop$ sha1sum Triage-Memory.mem
c95e8cc8c946f95a109ea8e47a6800de10a27abd  Triage-Memory.mem
```

Answer: c95e8cc8c946f95a109ea8e47a6800de10a27abd

What volatility profile is the most appropriate for this machine? (ex: Win10x86_14393)

In order to determine the volatility profile we should use for this memory dump, you can use the imageinfo plugin which performs a KDBG search:

```
remnux@remnux:~/Desktop$ vol.py -f Triage-Memory.mem imageinfo
Volatility Foundation Volatility Framework 2.6.1
/usr/local/lib/python2.7/dist-packages/volatility/plugins/community/YingLi/ssh_agent_key.py:12: CryptographyDeprecationWarning: Python 2 is no longer supported by the Pytho
ecated in cryptography, and will be removed in the next release.
  from cryptography.hazmat.backends.openssl import backend
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/remnux/Desktop/Triage-Memory.mem)
                      PAE type : No PAE
                           DTB : 0x187000L
                          KDBG : 0xf800029f80a0L
          Number of Processors : 2
     Image Type (Service Pack) : 1
                KPCR for CPU 0 : 0xfffff800029f9d00L
                KPCR for CPU 1 : 0xfffff880009ee000L
             KUSER_SHARED_DATA : 0xfffff78000000000L
           Image date and time : 2019-03-22 05:46:00 UTC+0000
     Image local date and time : 2019-03-22 01:46:00 -0400
```

Answer: Win7SP1x64

What was the process ID of notepad.exe?

To find the process ID of notepad.exe, we can utilise the pslist plugin and grep for the executable:

```
vol.py -f Triage-Memory.mem --profile=Win7SP1x64 pslist | grep notepad.exe
```

```
0xfffffa80054f9060 notepad.exe            3032   1432      1       60      1      0 2019-03-22 05:32:22 UTC+0000
```

Answer: 3032

Name the child process of wscript.exe.

The pstree plugin enables us to determine the parent child hierarchy of processes within the memory dump:

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

```
. 0xfffffa8004905620:hfs.exe                    3952   1432
.. 0xfffffa8005a80060:wscript.exe               5116   3952
... 0xfffffa8005a1d9e0:UWkpjFjDzM.exe           3496   5116
.... 0xfffffa8005bb0060:cmd.exe                 4660   3496
```

Answer: UWkpjFjDzM.exe

What was the IP address of the machine at the time the RAM dump was created?

In order to determine the local IP address, we can utilise the netscan plugin that enumerates all network artifacts within the dump:

```
remnux@remnux:~/Desktop$ vol.py -f Triage-Memory.mem --profile=Win7SP1x64 netscan
Volatility Foundation Volatility Framework 2.6.1
/usr/local/lib/python2.7/dist-packages/volatility/plugins/community/YingLi/ssh_age
ecated in cryptography, and will be removed in the next release.
  from cryptography.hazmat.backends.openssl import backend
Offset(P)          Proto    Local Address                  Foreign Address      St
0x13e057300        UDPv4    10.0.0.101:55736               *:*
```

Answer: 10.0.0.101

Based on the answer regarding the infected PID, can you determine the IP of the attacker?

Based on the process hierarchy for hfs.exe, we can assume that the infected process is UWkpjFjDzM.exe which we discovered earlier. If you examine the output of the netscan plugin and look for connections spawned by UWkpjFjDzM.exe, we can se the remote address of the connection (i.e., the attackers IP address):

```
10.0.0.106:4444      ESTABLISHED      3496      UWkpjFjDzM.exe
```

Answer: 10.0.0.106

How many processes are associated with VCRUNTIME140.dll?

For this question, we can utilise the dlllist plugin to enumerate all the DLLs loaded by each process. To reduce the output, we can grep for the DLL of interest like as follows:

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

```
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
```

As you can see, 5 processes are associated with VCRUNTIME140.dll.

Answer: 5

After dumping the infected process, what is its md5 hash?

In order to dump the infected process (UWkpjFjDzM.exe), we can utilise the procdump plugin like as follows:

```
vol.py -f Triage-Memory.mem --profile=Win7SP1x64 procdump -p 3496 -D .
```

Where -p specifies the PID and -D specifies the dump directory. We can then utilise the md5sum utility to generate the MD5 hash for the dumped process:

```
remnux@remnux:~/Desktop$ md5sum executable.3496.exe
690ea20bc3bdfb328e23005d9a80c290  executable.3496.exe
```

Answer: 690ea20bc3bdfb328e23005d9a80c290

What is the LM hash of Bob's account?

To find Bob's LM hash, we can utilise the hashdump plugin like as follows. This plugin allows us to retrieve password hashes stored in memory:

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

```
Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
Bob:1000:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
```

NTLM hashes are broken down into 4 parts, the username, relative identifier, LM hash, and NT hash, therefore, Bob's LM hash is aad3b435b51404eeaad3b435b51404ee.

Answer: aad3b435b51404eeaad3b435b51404ee

What memory protection constants does the VAD node at 0xfffffa800577ba10 have?

Each process is assigned a set of Virtual Address Descriptors (VADs) that describe the ranges of virtual memory address space reversed for the process. It also stored information about allocation type, memory protection, etc. The vadinfo plugin in Volatility enables us to find the VAD info for a process:

```
vol.py -f Triage-Memory.mem --profile=Win7SP1x64 vadinfo | grep -A 5 0xfffffa800577ba10
```

```
VAD node @ 0xfffffa800577ba10 Start 0x0000000000030000 End 0x0000000000033fff Tag Vad
Flags: NoChange: 1, Protection: 1
Protection: PAGE_READONLY
Vad Type: VadNone
ControlArea @fffffa8005687a50 Segment fffff8a000c4f870
NumberOfSectionReferences:          1 NumberOfPfnReferences:           0
```

Answer: PAGE_READONLY

What memory protection did the VAD starting at 0x00000000033c0000 and ending at 0x00000000033dffff have?

Follow the same process as the previous question:

```
vol.py -f Triage-Memory.mem --profile=Win7SP1x64 vadinfo | grep -A 5 "0x00000000033c0000 End 0x00000000033dffff"
```

```
VAD node @ 0xfffffa80052652b0 Start 0x00000000033c0000 End 0x00000000033dffff Tag VadS
Flags: CommitCharge: 32, PrivateMemory: 1, Protection: 24
Protection: PAGE_NOACCESS
Vad Type: VadNone
```

Answer: PAGE_NOACCESS

There was a VBS script that ran on the machine. What is the name of the script? (submit without file extension)

We can utilise the cmdline plugin that shows all the command-line arguments used by each process, and pipe the output to grep:

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

```
Command line : "C:\Windows\System32\wscript.exe" //B //NOLOGO %TEMP%\vhjReUDEuumrX.vbs
```

Answer: vhjReUDEuumrX

An application was run at 2019-03-07 23:06:58 UTC. What is the name of the program? (Include extension)

Within Windows, there is a forensic artifact called the ShimCache. ShimCache records the executable file name, file path, and last modification date and time. By analysing the ShimCache, you are able to find evidence of execution. Fortunately, Volatility has a ShimCache plugin that we can utilise:

```
vol.py -f Triage-Memory.mem --profile=Win7SP1x64 shimcache | grep "2019-03-07 23:06:58 UTC"
```

```
2019-03-07 23:06:58 UTC+0000   \??\C:\Program Files (x86)\Microsoft\Skype for Desktop\Skype.exe
```

At the specifies time, you can see that Skype.exe was executed.

Answer: Skype.exe

What was written in notepad.exe at the time when the memory dump was captured?

If you recall earlier, we determined the PID of notepad to be 3032. Let's dump this processes memory and inspect it:

```
vol.py -f Triage-Memory.mem --profile=Win7SP1x64 memdump -p 3032 -D .
```

```
remnux@remnux:~/Desktop$ strings -e l ./3032.dmp | grep flag
flag<REDBULL_IS_LIFE>
```

Answer: flag<REDBULL_IS_LIFE>

What is the short name of the file at file record 59045?

To find the short name of the file at the given record, we can utilise the mftparser plugin. For context, the MFT file stores information about all files on the filesystem and where they are located.

```
vol.py -f Triage-Memory.mem --profile=Win7SP1x64 mftparser | grep -A 15 "Record Number: 59045"
```

```
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
```

Answer: EMPLOY~1.XLS

This box was exploited and is running meterpreter. What was the infected PID?

Recall in question 8 how we dumped the infected process (PID 3496), if you enter this hash in VirusTotal you can see that it's tagged as meterpreter by several engines:

Answer: 3496