CyberDefenders: BlackEnergy Lab

The following writeup is for BlackEnergy Lab on CyberDefenders, it analysing a memory dump using volatility.

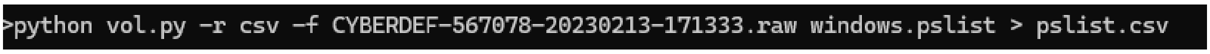
Scenario: A multinational corporation has been hit by a cyber attack that has led to the theft of sensitive data. The attack was carried out using a variant of the BlackEnergy v2 malware that has never been seen before. The company's security team has acquired a memory dump of the infected machine, and they want you, as a soc analyst, to analyse the dump to understand the attack scope and impact.

Which volatility profile would be best for this machine?

WinXPSP2x86

How many processes were running when the image was acquired?

We can use the pslist command like as follows:

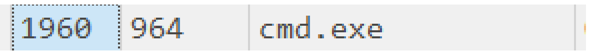


```
>python vol.py -r csv -f CYBERDEF-567078-20230213-171333.raw windows.pslist > pslist.csv
```

There were 19 processes running.

What is the process ID of cmd.exe?

In the output of the pslist plugin, we can see the PID of cmd.exe is 1960:



1960	964	cmd.exe

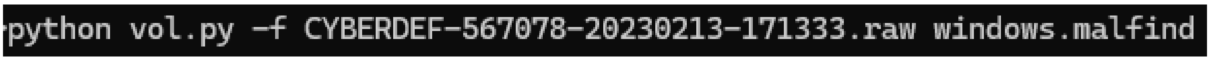
What is the name of the most suspicious process?

In the output of the pslist plugin, we can also see a very suspicious process called rootkit.exe:

```
rootkit.exe
```

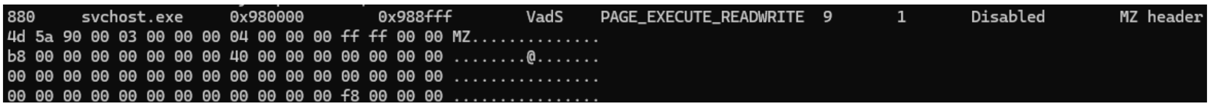
Which process shows the highest likelihood of code injection?

We can use the malfind plugin to look for injected code/DLLs:



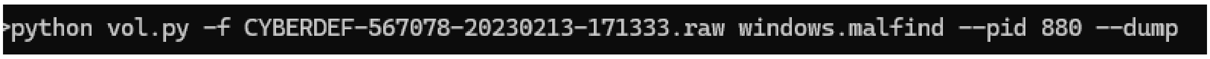
```
python vol.py -f CYBERDEF-567078-20230213-171333.raw windows.malfind
```

After looking through the output, we can see the PE header in svchost.exe and the VadS set to PAGE_EXECUTE_READWRITE, this immediately stands out to me and is also the answer:

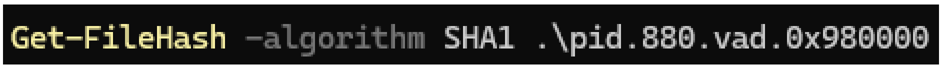


```
880     svchost.exe     0x980000        0x988fff        VadS    PAGE_EXECUTE_READWRITE  9       1       Disabled        MZ header
4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00 MZ..............
b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 ........@.......
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00 00 00 00 00 00 00 00 00 00 00 00 f8 00 00 00 ................
```

Let's confirm this by dumping this process:

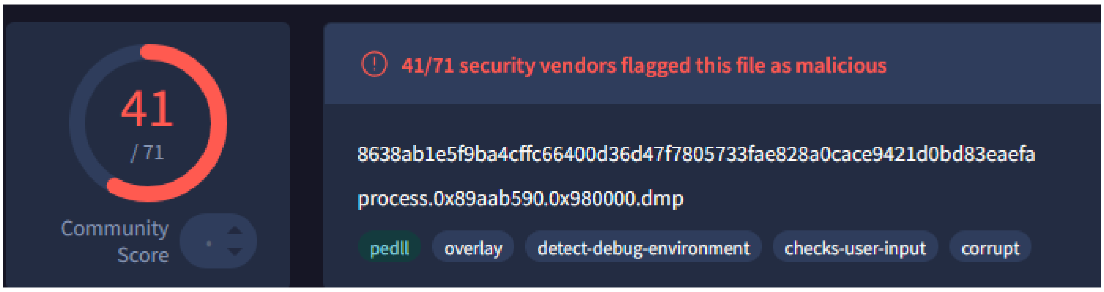


```
python vol.py -f CYBERDEF-567078-20230213-171333.raw windows.malfind --pid 880 --dump
```



```
Get-FileHash -algorithm SHA1 .\pid.880.vad.0x980000
```

If you enter the SHA1 hash into VirusTotal, you can see 41 detection which is extremely suspicious, confirming our original suspicions:



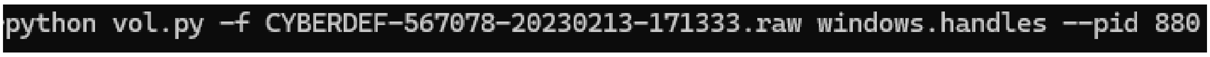
41/71 security vendors flagged this file as malicious

8638ab1e5f9ba4cffc66400d36d47f7805733fae828a0cace9421d0bd83eaefa

process.0x89aab590.0x980000.dmp

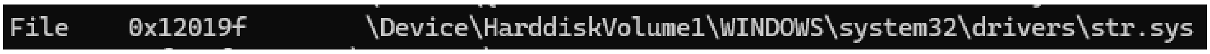
There is an odd file referenced in the recent process. Provide the full path of that file.

We can use the windows.handles plugin to look for any handles svchost has with files:



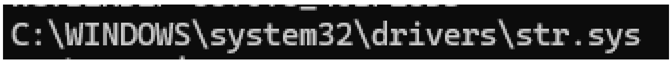
```
python vol.py -f CYBERDEF-567078-20230213-171333.raw windows.handles --pid 880
```

If you look through the output, you can find an interesting handle to str.sys:



```
File    0x12019f        \Device\HarddiskVolume1\WINDOWS\system32\drivers\str.sys
```

This is also in the strings of the file we dumped previousy:

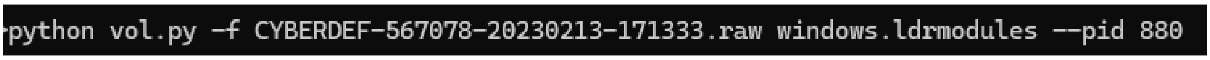


```
C:\WINDOWS\system32\drivers\str.sys
```

Therefore, the answer is C:\WINDOWS\system32\drivers\str.sys

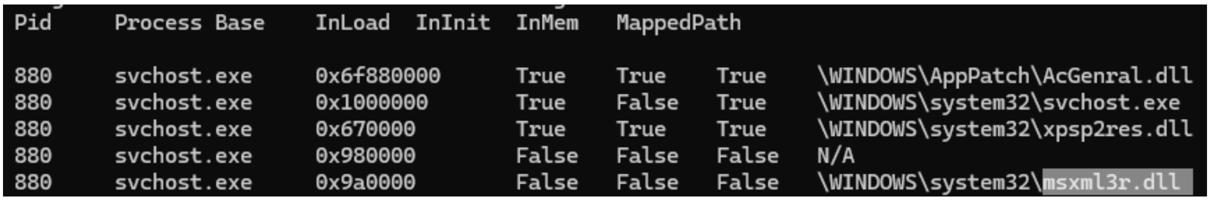
What is the name of the injected dll file loaded from the recent process?

We can use the ldrmodules plugin to list loaded modules in a particular process (in this instance, svchost.exe):



```
python vol.py -f CYBERDEF-567078-20230213-171333.raw windows.ldrmodules --pid 880
```

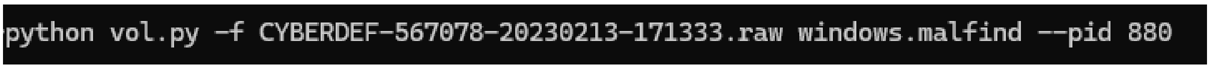
When looking at the output, msxml3r.dll stands out as very suspicious:



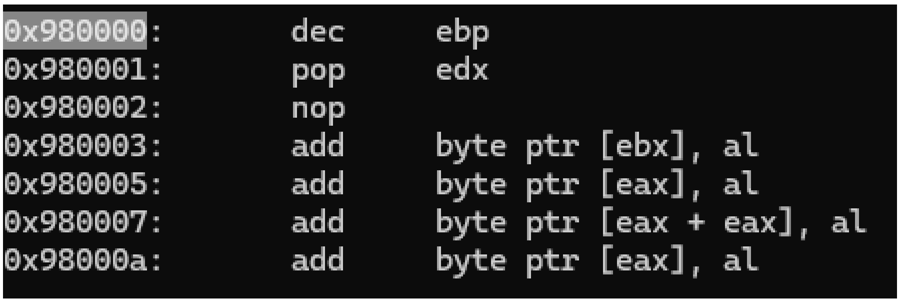
```
Pid     Process Base    InLoad  InInit  InMem   MappedPath

880     svchost.exe     0x6f880000      True    True    True    \WINDOWS\AppPatch\AcGenral.dll
880     svchost.exe     0x1000000       True    False   True    \WINDOWS\system32\svchost.exe
880     svchost.exe     0x670000        True    True    True    \WINDOWS\system32\xpsp2res.dll
880     svchost.exe     0x980000        False   False   False   N/A
880     svchost.exe     0x9a0000        False   False   False   \WINDOWS\system32\msxml3r.dll
```

This is mainly due to how InLoad, InInit, and InMem are all set to False. This means that msxml3r.dll is not properly loaded, initalised, or present in memory. This is very unusual for a legitimate DLL; therefore we can infer that this one is malicious.

What is the base address of the injected dll?



```
python vol.py -f CYBERDEF-567078-20230213-171333.raw windows.malfind --pid 880
```



```
0x980000:       dec     ebp
0x980001:       pop     edx
0x980002:       nop
0x980003:       add     byte ptr [ebx], al
0x980005:       add     byte ptr [eax], al
0x980007:       add     byte ptr [eax + eax], al
0x98000a:       add     byte ptr [eax], al
```

The base address is 0x980000.

This was the most difficult memory forensics challenge I have ever done, nonetheless I really enjoyed the entire process, especially looking for injected code. If you are relatively new to volatility, I highly recommend completing this room.