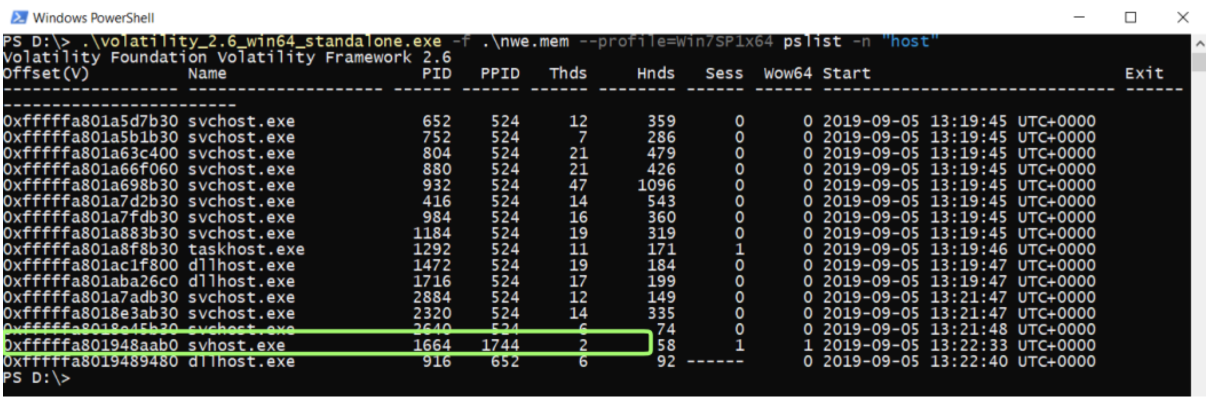
Ch.5 Malware Detection and Analysis with Windows Memory Forensics

Malware detection in Memory Methodology

1. **Searching for malicious processes**
2. **Analyzing command-line arguments**
3. **Examining network connections**
4. **Detecting injections in process memory**
5. **Looking for evidence of persistence**
6. **Creating timelines**

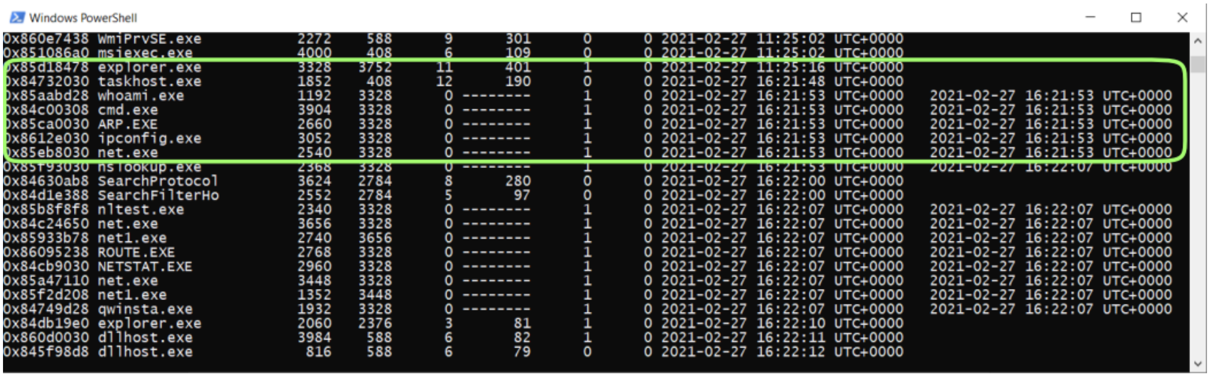
1- Searching for malicious processes

1. **you need to learn about system processes.**
2. **try to find a way to masquerade as a system process or, in the worst-case scenario, take advantage of a legitimate process.**



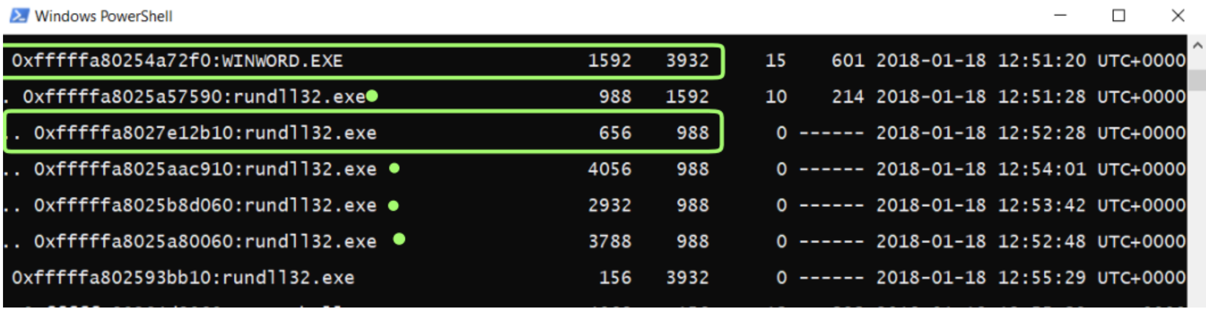
```
Windows PowerShell
PS D:\> .\volatility_2.6_win64_standalone.exe -f .\nwe.mem --profile=Win7SP1x64 pslist -n "host"
Volatility Foundation Volatility Framework 2.6
Offset(V)          Name                    PID   PPID   Thds     Hnds   Sess  Wow64 Start                          Exit
------------------ -------------------- ------ ------ ------ -------- ------ ------ ------------------------------ ------
------------------------
0xfffffa801a5d7b30 svchost.exe             652    524     12      359      0      0 2019-09-05 13:19:45 UTC+0000
0xfffffa801a5b1b30 svchost.exe             752    524      7      286      0      0 2019-09-05 13:19:45 UTC+0000
0xfffffa801a63c400 svchost.exe             804    524     21      479      0      0 2019-09-05 13:19:45 UTC+0000
0xfffffa801a66f060 svchost.exe             880    524     21      426      0      0 2019-09-05 13:19:45 UTC+0000
0xfffffa801a698b30 svchost.exe             932    524     47     1096      0      0 2019-09-05 13:19:45 UTC+0000
0xfffffa801a7d2b30 svchost.exe             416    524     14      543      0      0 2019-09-05 13:19:45 UTC+0000
0xfffffa801a7fdb30 svchost.exe             984    524     16      360      0      0 2019-09-05 13:19:45 UTC+0000
0xfffffa801a883b30 svchost.exe            1184    524     19      319      0      0 2019-09-05 13:19:45 UTC+0000
0xfffffa801a8f8b30 taskhost.exe           1292    524     11      171      1      0 2019-09-05 13:19:46 UTC+0000
0xfffffa801ac1f800 dllhost.exe            1472    524     19      184      0      0 2019-09-05 13:19:47 UTC+0000
0xfffffa801aba26c0 dllhost.exe            1716    524     17      199      0      0 2019-09-05 13:19:47 UTC+0000
0xfffffa801a7adb30 svchost.exe            2884    524     12      149      0      0 2019-09-05 13:21:47 UTC+0000
0xfffffa8018e3ab30 svchost.exe            2320    524     14      335      0      0 2019-09-05 13:21:47 UTC+0000
0xfffffa8018e45b30 svchost.exe            2640    524      6       74      0      0 2019-09-05 13:21:48 UTC+0000
0xfffffa801948aab0 svhost.exe             1664   1744      2       58      1      1 2019-09-05 13:22:33 UTC+0000
0xfffffa8019489480 dllhost.exe             916    652      6       92 ------      0 2019-09-05 13:22:40 UTC+0000
PS D:\>
```

3. **While some malicious programs hide behind the mask of legitimate processes, `whoami.exe`, `ipconfig.exe`, `netstat.exe`, and more. These utilities can be used by system administrators or advanced users to check the settings and configure the network.**



Detecting abnormal behavior

- **Abnormal behavior can result in many things. For some processes, it will be atypical to make network connections, and for others, it will be atypical to spawn new processes or access certain filesystem objects.**



1. **the `WINWORD.EXE` process spawns a child process, `rundll32.exe` , this behavior could be the result of macros embedded inside a document that has been opened by a user**

> 💡 **Rundll32 loads and runs 32-bit dynamic-link libraries (DLLs) that can be used for directly invoking specified functions**

1. **find all files used by the proccess using `handle` plugin.**
2. **`filescan` to get physical offset where the document is located.**
3. **dump the files using `dumpfiles` and then go to any threat intel like VT (Virus Total).**
4. **analyze the document with oletools utility `pip3.exe install -U oletools`**

- `olevba --reveal 'malicioius.doc'`

```
Windows PowerShell
+----------+--------------------+---------------------------------------------+
|Type      |Keyword             |Description                                  |
+----------+--------------------+---------------------------------------------+
|AutoExec  |AutoOpen            |Runs when the Word document is opened        |
|AutoExec  |Auto_Open           |Runs when the Excel Workbook is opened       |
|AutoExec  |Workbook_Open       |Runs when the Excel Workbook is opened       |
|Suspicious|Environ             |May read system environment variables        |
|Suspicious|Lib                 |May run code from a DLL                      |
|Suspicious|VirtualAllocEx      |May inject code into another process         |
|Suspicious|WriteProcessMemory  |May inject code into another process         |
|Suspicious|Base64 Strings      |Base64-encoded strings were detected, may be |
|          |                    |used to obfuscate strings (option --decode to|
|          |                    |see all)                                     |
|Suspicious|VBA obfuscated      |VBA string expressions were detected, may be |
|          |Strings             |used to obfuscate strings (option --decode to|
|          |                    |see all)                                     |
|IOC       |rundll32.exe        |Executable file name                         |
|IOC       |undll32.exe         |Executable file name (obfuscation: VBA       |
|          |                    |expression)                                  |
|VBA string|%ProgramW6432%      |(Environ("ProgramW6432"))                    |
|VBA string|%windir%\\SysWOW64\ |Environ("windir") &                          |
|          |undll32.exe         |"\\SysWOW64\\rundll32.exe"                   |
|VBA string|%windir%\\System32\ |Environ("windir") &                          |
|          |undll32.exe         |"\\System32\\rundll32.exe"                   |
+----------+--------------------+---------------------------------------------+
MACRO SOURCE CODE WITH DEOBFUSCATED VBA STRINGS (EXPERIMENTAL):
```

2- Analyzing command-line arguments

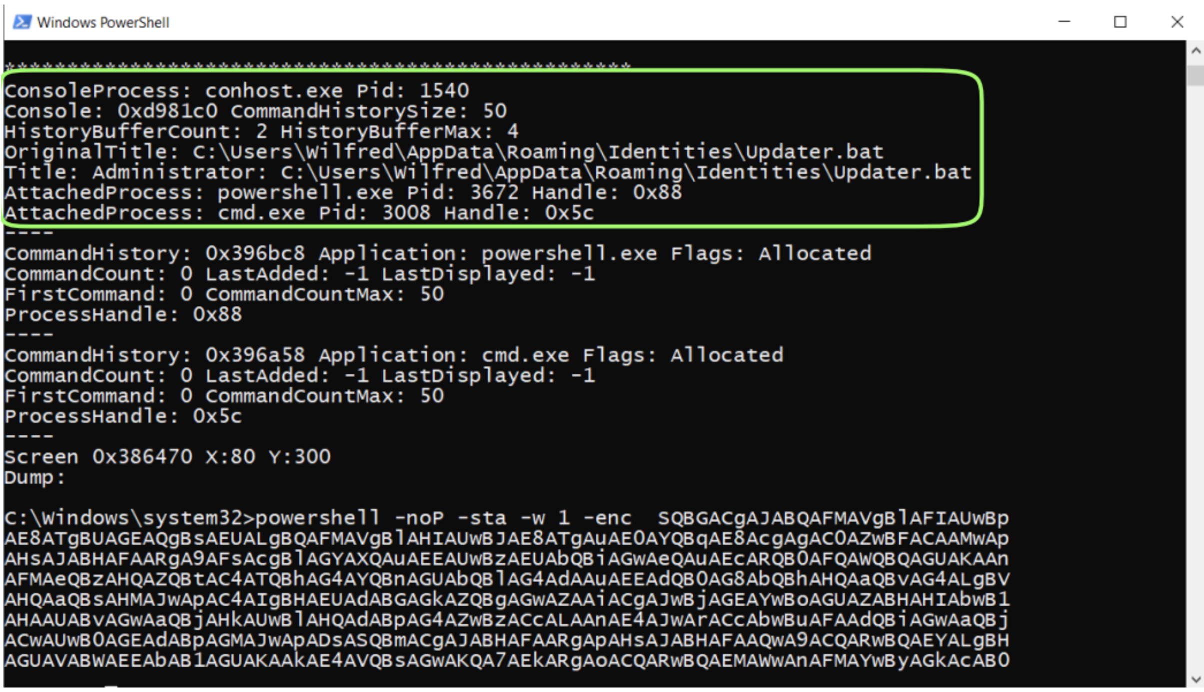
- **Analyzing command-line arguments is very important because it allows you to check the location from which the executable was run and the arguments passed to it.**

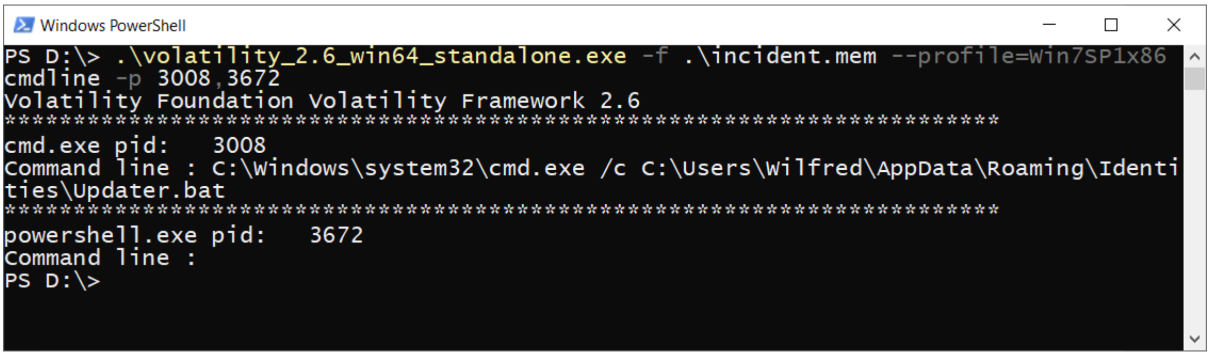
1. `vol.py -f file.mem --profile=win7SP1x64 pstree -v` **this will show each process with it's detailed commend line to start a particular program.**
2. `vol.py -f file.mem --profile=win7SP1x64 cmdline -n '(cmd|powershell|psexec)'` **{-n : for grep to keyword}**

```
Select Windows PowerShell
PS D:\> .\volatility_2.6_win64_standalone.exe -f .\DFA\Inside.vmem --profile=Win10x64_14393
cmdline -n '(cmd|powershell|psexec)'
Volatility Foundation Volatility Framework 2.6
************************************************************************
MpCmdRun.exe pid:   4424
Command line :
************************************************************************
cmd.exe pid:   7288
Command line : "C:\Windows\system32\cmd.exe"
************************************************************************
PsExec64.exe pid:   6748
Command line : PsExec64.exe  \\win7 -u hack\max -p P@sswOrd -accepteula cmd.exe
************************************************************************
cmd.exe pid:   4240
PS D:\>
```

3. **get the commands stored in memory through** `cmdscan`**.**
4. **you can use** `vol.py -f file.mem --profile=win7SP1x64 yarascan -y rule.yar` **with external yara rule to detect malicious commands**

5. `consoles` plugin allows you to get data regarding the commands executed by different command-line interpreters: cmd, PowerShell, the Python shell, and the Perl shell.





🧠 **we can conclude that PowerShell, with all of its options, in the content of the same Updater.bat file, which is executed through cmd.**

3- Examining network connections

- **Aside from atypical initiators, there are some processes that we have to keep an eye on. These include `cmd.exe` and `powershell.exe`. If you have detected connections established by these processes, be sure to check the IP addresses specified in the Foreign Address field.**

- **there are also default protocols used in tools such as port scanners or post-exploitation frameworks.**

Port	Tool
81,9001	TOR project applications and TOR
689	Nmap port and vulnerability scanner
1241	Nessus vulnerability scanner
3899,4899	RAdmin
3790	Metasploit
4444	Meterpreter reverse shell
50050	Cobalt Strike Team Server

Detecting injections in process memory

- **the injection of Dynamic link Libraries (DLLs) is one of the methods used to execute arbitrary code in the address space of a legitimate process.**
- **There are two main types of DLL injections: remote and reflective:**

1- Remote DLL injections Process

1. **The malicious process gets SeDebugPrivilege, which allows it to act as a debugger and gain read and write access to the address space of other processes.**
2. **the malicious process opens a handle for the target process, accesses its address space, and writes the full path to the malicious library inside it.**
3. **Create a new thread to load the malicious DLL from the disk using Windows API functions into target process's address space.**

> 🧠 **A thread refers to a single sequential flow of activities being executed in a process.**

4. **Delete the path to the malicious DLL from the target process' memory.**
5. **Close the handle to the target process.**

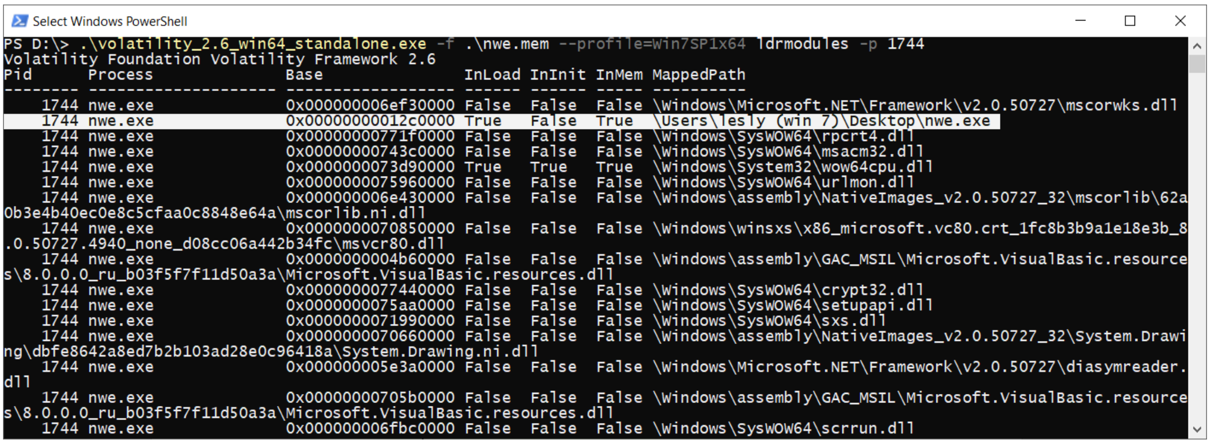
- **we can use Volatility plugins such as `dlllist` and `ldrmodules` to detect this.**

Information about the libraries used by the process is stored in three different lists:

1. **LoadOrderList organizes the order in which modules are loaded into a process.**
2. **MemoryOrderList organizes the order in which modules appear in the process' virtual memory.**
3. **InitOrderList organizes the order in which the DllMain function is executed.**

> 💡 **The dlllist plugin only works with LoadOrderList , sometimes, malicious libraries can be unlinked from this list to hide their presence.**

- **`ldrmodules` plugin it's outputs information from all three lists but also provides data regarding the presence of this or that library in each of the lists:**



```
Select Windows PowerShell
PS D:\> .\volatility_2.6_win64_standalone.exe -f .\nwe.mem --profile=Win7SP1x64 ldrmodules -p 1744
Volatility Foundation Volatility Framework 2.6
Pid      Process              Base               InLoad InInit InMem MappedPath
-------- -------------------- ------------------ ------ ------ ----- ----------
    1744 nwe.exe              0x000000006ef30000 False  False  False \Windows\Microsoft.NET\Framework\v2.0.50727\mscorwks.dll
    1744 nwe.exe              0x00000000012c0000 True   False  True  \Users\lesly (win 7)\Desktop\nwe.exe
    1744 nwe.exe              0x00000000771f0000 False  False  False \Windows\SysWOW64\rpcrt4.dll
    1744 nwe.exe              0x00000000743c0000 False  False  False \Windows\SysWOW64\msacm32.dll
    1744 nwe.exe              0x0000000073d90000 True   True   True  \Windows\System32\wow64cpu.dll
    1744 nwe.exe              0x0000000075960000 False  False  False \Windows\SysWOW64\urlmon.dll
    1744 nwe.exe              0x000000006e430000 False  False  False \Windows\assembly\NativeImages_v2.0.50727_32\mscorlib\62a
0b3e4b40ec0e8c5cfaa0c8848e64a\mscorlib.ni.dll
    1744 nwe.exe              0x0000000070850000 False  False  False \Windows\winsxs\x86_microsoft.vc80.crt_1fc8b3b9a1e18e3b_8
.0.50727.4940_none_d08cc06a442b34fc\msvcr80.dll
    1744 nwe.exe              0x0000000004b60000 False  False  False \Windows\assembly\GAC_MSIL\Microsoft.VisualBasic.resource
s\8.0.0.0_ru_b03f5f7f11d50a3a\Microsoft.VisualBasic.resources.dll
    1744 nwe.exe              0x0000000077440000 False  False  False \Windows\SysWOW64\crypt32.dll
    1744 nwe.exe              0x0000000075aa0000 False  False  False \Windows\SysWOW64\setupapi.dll
    1744 nwe.exe              0x0000000071990000 False  False  False \Windows\SysWOW64\sxs.dll
    1744 nwe.exe              0x0000000070660000 False  False  False \Windows\assembly\NativeImages_v2.0.50727_32\System.Drawi
ng\dbfe8642a8ed7b2b103ad28e0c96418a\System.Drawing.ni.dll
    1744 nwe.exe              0x000000005e3a0000 False  False  False \Windows\Microsoft.NET\Framework\v2.0.50727\diasymreader.
dll
    1744 nwe.exe              0x00000000705b0000 False  False  False \Windows\assembly\GAC_MSIL\Microsoft.VisualBasic.resource
s\8.0.0.0_ru_b03f5f7f11d50a3a\Microsoft.VisualBasic.resources.dll
    1744 nwe.exe              0x000000006fbc0000 False  False  False \Windows\SysWOW64\scrrun.dll
```

> 💡 **You can detect the libraries that have been unlinked. These libraries will show False in the InLoad column and True in the other columns.**

- **use `dlldump` or `dumpfiles` to dump DLLs.**

`vol.py  -f  file.mem --profile=win7SP1x64 dlldump -p 1072 -D .\output\`

- **To quickly calculate the hash of the DLLs, you can use the following PowerShell command:** `Get-ChildItem .\ | ForEach-Object -Process {Get-FileHash -Algorithm SHA1 $_.Name}`

2- Reflective DLL injections

- **The library can be downloaded over the network and immediately injected into process memory. Another feature of this method is the use of a reflective loader, which is embedded in the library itself, instead of the standard Windows loader**

1. **Get privileges and open a handle to the target process.**
2. **Allocate memory in the target process and write the malicious DLL there.**
3. **Create a new thread to invoke the reflective loader.**
4. **Close the handle to the target process**

- **when using this technique (just as with packers), a page with the EXECUTE_READWRITE protection is created in the target process memory.**
- `malfind` **plugin allows you to find such pages in process memory and check them for executable file headers or correct CPU instructions.**

`vol.py  -f  file.mem --profile=win7SP1x64 malfind -p 1072 -D .\output\` **{-D : to dump the injection code section}**

3- Portable executable injections

1. **Get privileges and open a handle to the target process.**
2. **Allocate memory in the target process and write malicious code there.**
3. **Create a new thread to run the injected code.**
4. **Close the handle to the target process.**

> 💡 **please note that** `malfind` **only analyzes private memory regions with read, write, and execute access. This means the detectability of this plugin can be bypassed.**

> https://github.com/JPCERTCC/aa-tools/blob/master/cobaltstrikescan.py **{external plugin}**

```
Windows PowerShell
PS D:\> .\volatility_2.6_win64_standalone.exe --plugins=./plugins -f
.\Windows7x64.vmem --profile=Win7SP1x64 cobaltstrikescan
Volatility Foundation Volatility Framework 2.6
Name                 PID      Data VA
-------------------- -------- ------------------
OUTLOOK.EXE              3932 0x0000000003c70000
rundll32.exe              988 0x0000000002980000
powershell.exe           3876 0x0000000005a40000
PS D:\>
```

- **This means that in the memory of these processes, you can find its configurations, where useful parameters such as the C2 IP addresses are located.**

4- Process Hollowing

hollow process : injection is to create a new instance of a legitimate process in the SUSPEND state and overwrite the address space occupied by its executable code with malicious code.

- **Two methods can be used to detect process hollowing :**

1. **The first one involves comparing PEB and Virtual Address Descriptor (VAD) structures and searching for inconsistencies through `psinfo` , `hollowfind` plugins —>** `vol.py -f file.mem --profile=win7SP1x64 psinfo/hollowfind -p 1664`

> 💡 **what will we see with process hollowing? Well, the information taken from the PEB will match the process used as a container, but the VAD structure will no longer have a file mapped to this memory region.**

2. **we can detect process hollowing through `ldrmodules` plugin .**

> 💡 **n the case of process hollowing, the flags (True False True) in (InLoad InInit InMem) will remain, but the path to the executable file will be missing.**

5- Process Doppelgänging

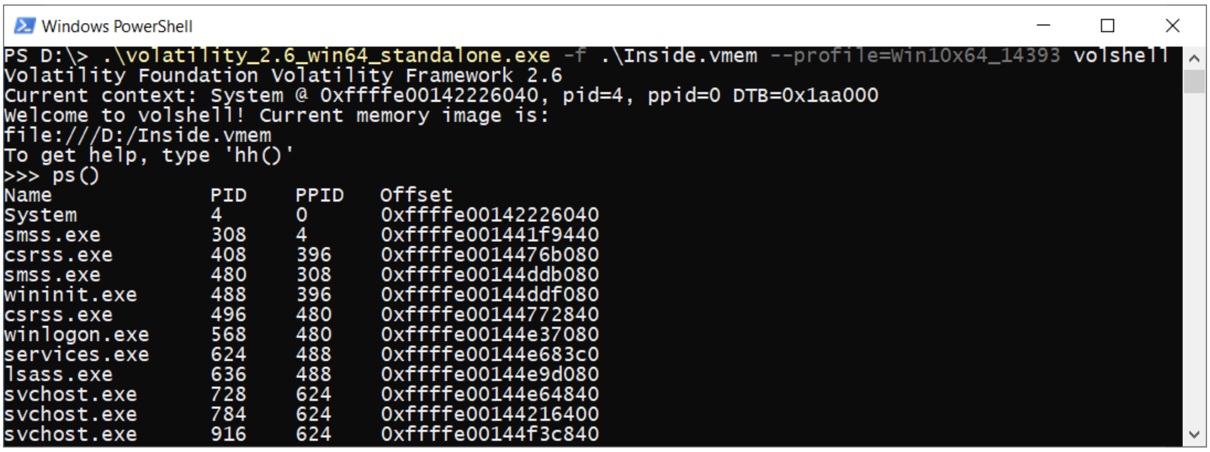
- **This technique is based on the use of NTFS transactions.**

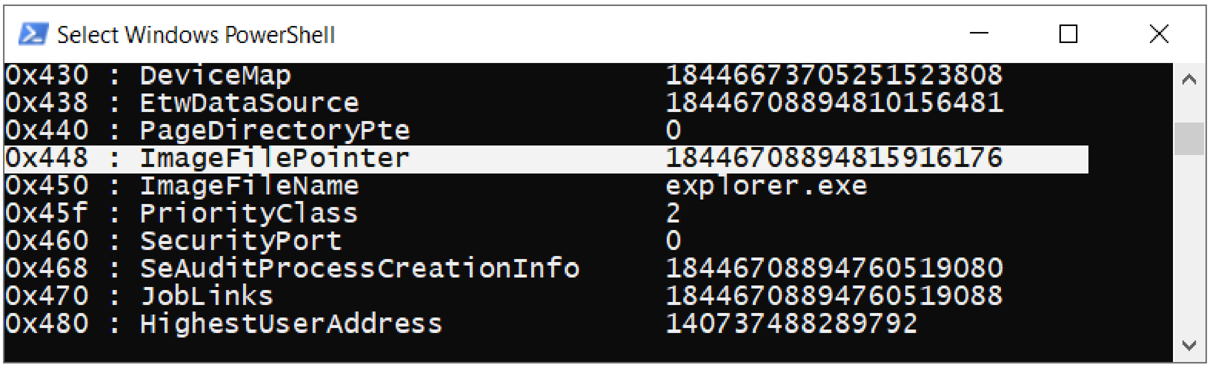
1. **Create a transaction and open a clean transacted file.**
2. **Overwrite the transacted file with malicious code.**
3. **Create a memory section that points to the transacted file.**
4. **Roll back the transaction (this will remove all the traces of the transacted file from the filesystem but not the memory section where the malicious code was mapped).**
5. **Create objects, process and thread objects; set the start address of the thread to the entry point of the malicious code.**
6. **Create process parameters and copy them to the newly created process' address space.**
7. **Run the doppelgänged process.**

- **The use of this technique is quite difficult to detect. For systems older than Windows 10, you can check the File_Object associated with the suspicious process. If write access for this file is enabled, that could potentially be Process Doppelgänging.**

💡 **For Windows 10 systems, check _EPROCESS of the suspected process through `vollshell` plugin , if the ImageFilePointer is set to NULL , there is process Doppelgänging**

```
ps()
dt('_EPROCESS', {process offset})
```





Looking for evidence of persistence

1- Boot or Logon Autostart Execution technique

- **HKLM \SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon**
- **HKLM\Software\Microsoft\Windows\CurrentVersion\Run**
- **HKLM\Software\Microsoft\Windows\CurrentVersion\RunOnce**
- **HKCU\Software\Microsoft\Windows\CurrentVersion\Run**
- **HKCU\Software\Microsoft\Windows\CurrentVersion\RunOnce**
- **HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options**
- **HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\SilentProcessExit**

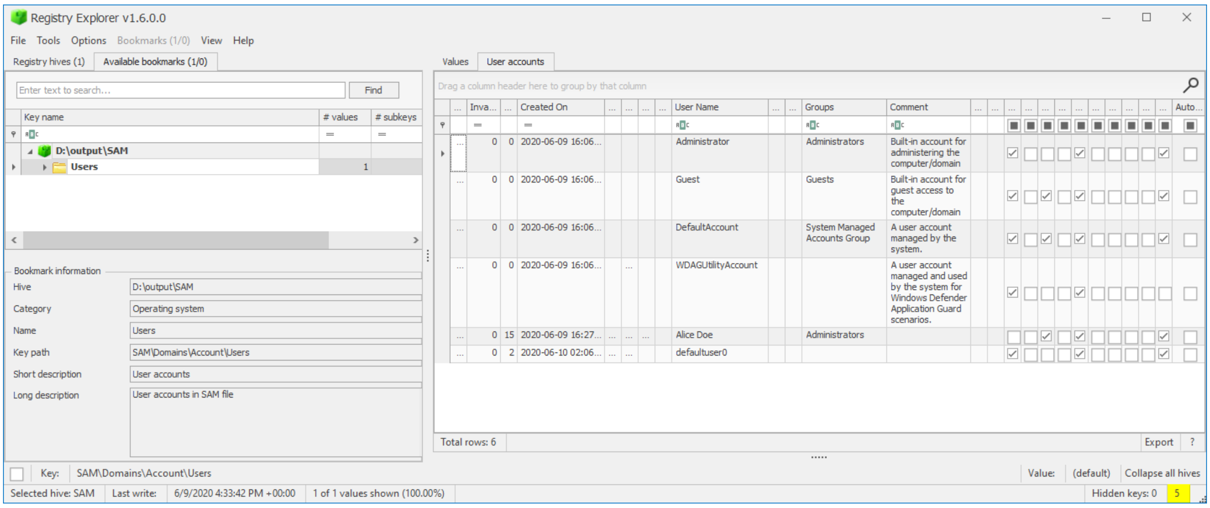
1. **you can start by examining the output of the `handles` plugin with the `-t Key` option, which shows all of the registry keys used by this process: `vol.py -f file.mem --profile=win7SP1x64 handles -p 1744 -t key --silent`**

2. use `prinkey` plugin : `vol.py -f file.mem --profile=win7SP1x64 prinkey " Software\Microsoft\Windows\CurrentVersion\Run "`

2- Create Account technique

- **This technique is often used by ransomware operators, as it is excellent for maintaining access to compromised systems.**

1. **it is best to use the Registry Explorer tool and the bookmarks tab (click on Bookmarks and then Users)**



2. **searching in event logs, will be a good choice through `filescan` on memory and search for security.evtx log , and dump it and open in event viewer , Additional information regarding creating and enabling a user can be found in the 4720 and 4722 events.**

3- Create or Modify System Process

- **attackers install a new service that should run an executable file on disk or execute scripts.**

1. **The event ID of 7045: A service was installed in the system. When analyzing such events, you should pay attention to the name and location of the executable**
2. `svcscan` **plugin to get information about the running services, service names, types, states, binary paths, and more.**
3. `autoruns` **plugin developed by the community, which collects information not only about the services but also the various registry keys that could potentially be used for persistence.**

4- Scheduled task

- **scheduled tasks is stored in several locations:**

1. **C:\Windows\System32\Tasks : Here, you can find XML files with task descriptions.**
2. **Microsoft-Windows-TaskScheduler%4Operational.evtx , analyze event ID 106, which is related to the creation of a new task.**
3. **extract SOFTWARE form memory and run `rigripper` and search for taskcache keyword.**

Creating timelines

- **You find out details about what happened to the target system during a certain period of time, reconstruct the actions of the attackers step by step.**

1- Filesystem-based timelines

1. **For NTFS, this file would be, for example, the Master File Table ($MFT), through `mftparser` plugin, which collects all $MFT entries from memory.**

```
vol.py  -f  file.mem --profile=win7SP1x64 mftparser --output=body --output-file=.\output\body.txt
```

2. **convert body.txt to timeline using `perl` and `mactime.pl` form TheSleuthKit**

```
PS D:\> C:\Strawberry\perl\bin\perl.exe .\sleuthkit-4.10.2- win32\bin\mactime.pl -b .\output\body.txt > .\output\timeline. txt
```

> https://strawberryperl.com {perl}

> https://www.sleuthkit.org/sleuthkit/download.php {TheSleuthKit}

3. **OR you can Dump $MFT form memory and parse it using EricZimmerman.**

2- Memory-based timelines

1. `vol.py  -f  file.mem --profile=win7SP1x64 timeliner > timeline.txt`
2. **You cab use Redline to build timeline based on the data from memory dumps.**