

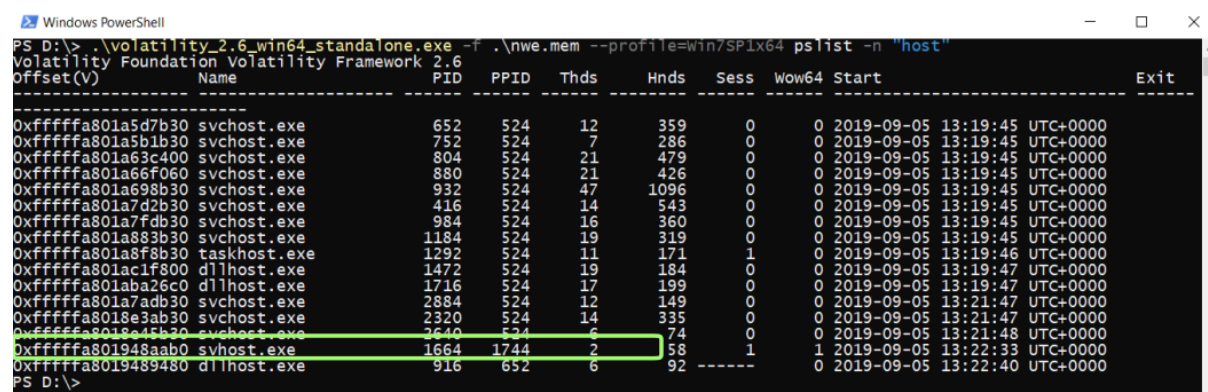
# Ch.5 Malware Detection and Analysis with Windows Memory Forensics

## 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.



```
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
-----
0xfffffffffa801a5d7b30  svchost.exe  652   524    12    359    0     0  2019-09-05 13:19:45 UTC+0000
0xfffffffffa801a5b1b30  svchost.exe  752   524     7    286    0     0  2019-09-05 13:19:45 UTC+0000
0xfffffffffa801a63c400  svchost.exe  804   524    21    479    0     0  2019-09-05 13:19:45 UTC+0000
0xfffffffffa801a66f060  svchost.exe  880   524    21    426    0     0  2019-09-05 13:19:45 UTC+0000
0xfffffffffa801a698b30  svchost.exe  932   524    47   1096    0     0  2019-09-05 13:19:45 UTC+0000
0xfffffffffa801a7d2b30  svchost.exe  416   524    14    543    0     0  2019-09-05 13:19:45 UTC+0000
0xfffffffffa801a7fdb30  svchost.exe  984   524    16    360    0     0  2019-09-05 13:19:45 UTC+0000
0xfffffffffa801a883b30  svchost.exe  1184  524    19    319    0     0  2019-09-05 13:19:45 UTC+0000
0xfffffffffa801a8f8b30  taskhost.exe  1292  524    11    171    0     0  2019-09-05 13:19:46 UTC+0000
0xfffffffffa801ac1f800  dllhost.exe  1472  524    19    184    0     0  2019-09-05 13:19:47 UTC+0000
0xfffffffffa801aba26c0  dllhost.exe  1716  524    17    199    0     0  2019-09-05 13:19:47 UTC+0000
0xfffffffffa801a7adb30  svchost.exe  2884  524    12    149    0     0  2019-09-05 13:21:47 UTC+0000
0xfffffffffa8018e3ab30  svchost.exe  2320  524    14    335    0     0  2019-09-05 13:21:47 UTC+0000
0xfffffffffa8018e4fb30  svchost.exe  2640  524     6     74    0     0  2019-09-05 13:21:48 UTC+0000
0xfffffffffa801948aab0  svchost.exe  1664  1744     2     58    1     1  2019-09-05 13:22:33 UTC+0000
0xfffffffffa8019489480  dllhost.exe  916   652     6     92    0     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.

Windows PowerShell									
0x86067438	wmiprvse.exe	2272	588	9	301	0	0	2021-02-27	11:25:02 UTC+0000
0x851086a0	msiexec.exe	4000	408	6	109	0	0	2021-02-27	11:25:02 UTC+0000
0x85d18478	explorer.exe	3928	3752	11	101	1	0	2021-02-27	11:25:16 UTC+0000
0x84732030	taskhost.exe	1852	408	12	190	0	0	2021-02-27	16:21:48 UTC+0000
0x85aab28	whoami.exe	1192	3328	0	-----	1	0	2021-02-27	16:21:53 UTC+0000
0x84c00308	cmd.exe	3904	3328	0	-----	1	0	2021-02-27	16:21:53 UTC+0000
0x85ca0030	ARP.EXE	2660	3328	0	-----	1	0	2021-02-27	16:21:53 UTC+0000
0x8612e030	ipconfig.exe	3052	3328	0	-----	1	0	2021-02-27	16:21:53 UTC+0000
0x85eb8030	net.exe	2540	3328	0	-----	1	0	2021-02-27	16:21:53 UTC+0000
0x85f33030	nslookup.exe	2368	3328	0	-----	1	0	2021-02-27	16:21:53 UTC+0000
0x84630ab8	SearchProtocol	3624	2784	8	280	0	0	2021-02-27	16:22:00 UTC+0000
0x84d1e388	SearchFilterHo	2552	2784	5	97	0	0	2021-02-27	16:22:00 UTC+0000
0x85b8f8f8	nltest.exe	2340	3328	0	-----	1	0	2021-02-27	16:22:07 UTC+0000
0x84c24650	net.exe	3656	3328	0	-----	1	0	2021-02-27	16:22:07 UTC+0000
0x85933b78	net1.exe	2740	3656	0	-----	1	0	2021-02-27	16:22:07 UTC+0000
0x86095238	ROUTE.EXE	2768	3328	0	-----	1	0	2021-02-27	16:22:07 UTC+0000
0x84cb9030	NETSTAT.EXE	2960	3328	0	-----	1	0	2021-02-27	16:22:07 UTC+0000
0x85a47110	net.exe	3448	3328	0	-----	1	0	2021-02-27	16:22:07 UTC+0000
0x85f2d208	net1.exe	1352	3448	0	-----	1	0	2021-02-27	16:22:07 UTC+0000
0x84749d28	qwinsta.exe	1932	3328	0	-----	1	0	2021-02-27	16:22:07 UTC+0000
0x84db19e0	explorer.exe	2060	2376	3	81	1	0	2021-02-27	16:22:10 UTC+0000
0x860d0030	dllhost.exe	3984	588	6	82	1	0	2021-02-27	16:22:11 UTC+0000
0x845f98d8	dllhost.exe	816	588	6	79	0	0	2021-02-27	16:22:12 UTC+0000

## 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.

Windows PowerShell									
0xfffffa80254a72f0:	WINWORD.EXE	1592	3932	15	601	2018-01-18	12:51:20	UTC+0000	
0xfffffa8025a57590:	rundll32.exe	988	1592	10	214	2018-01-18	12:51:28	UTC+0000	
0xfffffa8027e12b10:	rundll32.exe	656	988	0	-----	2018-01-18	12:52:28	UTC+0000	
0xfffffa8025aac910:	rundll32.exe	4056	988	0	-----	2018-01-18	12:54:01	UTC+0000	
0xfffffa8025b8d060:	rundll32.exe	2932	988	0	-----	2018-01-18	12:53:42	UTC+0000	
0xfffffa8025a80060:	rundll32.exe	3788	988	0	-----	2018-01-18	12:52:48	UTC+0000	
0xfffffa802593bb10:	rundll32.exe	156	3932	0	-----	2018-01-18	12:55:29	UTC+0000	

- 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

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

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 Strings	VBA string expressions were detected, may be 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\undll32.exe	Environ("windir") & "\\SysWOW64\\rundll32.exe"
VBA string	%windir%\System32\undll32.exe	Environ("windir") & "\\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.

- `vol.py -f file.mem --profile=win7SP1x64 pstree -v` this will show each process with it's detailed command line to start a particular program.
- `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@ssw0rd -accepteula cmd.exe
*****
cmd.exe pid: 4240
PS D:\>

```

- get the commands stored in memory through `cmdscan` .
- 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.

```
Windows PowerShell

*****
ConsoleProcess: conhost.exe Pid: 1540
Console: 0xd981c0 CommandHistorySize: 50
HistoryBufferCount: 2 HistoryBufferMax: 4
OriginalTitle: C:\Users\wilfred\AppData\Roaming\Identities\Updater.bat
Title: Administrator: C:\Users\wilfred\AppData\Roaming\Identities\Updater.bat
AttachedProcess: powershell.exe Pid: 3672 Handle: 0x88
AttachedProcess: cmd.exe Pid: 3008 Handle: 0x5c
-----
CommandHistory: 0x396bc8 Application: powershell.exe Flags: Allocated
CommandCount: 0 LastAdded: -1 LastDisplayed: -1
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x88
-----
CommandHistory: 0x396a58 Application: cmd.exe Flags: Allocated
CommandCount: 0 LastAdded: -1 LastDisplayed: -1
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x5c
-----
Screen 0x386470 X:80 Y:300
Dump:

C:\windows\system32>powershell -noP -sta -w 1 -enc SQBGACgAJABQAFMAVgB1AFIAUwBp
AE8ATgBUAGEAQgBsAEUALgBQAFMAVgB1AHIAUwBjAE8ATgAuAE0AYQBqAE8AcgAgAC0AZwBFACAAMwAp
AHSaJABHAFaARGa9AFsAcgB1AGYAXQAUAEeAUwBzAEUABQBjAGWAEQAUAECARQB0AFQAWQBQAGUAKAAh
AFMAeQBzAHQAZQBtAC4ATQBhAG4AYQBnAGUAbQB1AG4AdAAUAEeAdQB0AG8AbQBhAHQAaQBvAG4ALgBV
AHQAaQBzAHMAJwApAC4AIgBHAEUAdABGAGkAZQBgAGwAZAA1ACgAJwBjAGEAYwBoAGUAZABHAIAbwB1
AHAAUABvAGWAaQBjAHkAUwB1AHQAdABpAG4AZwBzACCALAAhAE4AJwArAccAbwBuAFAdQBjAGWAaQBj
ACwAUwB0AGEAdABpAGMAJwApADSASQBmACgAJABHAFaARGApAHSaJABHAFaAQwA9ACQARwBQAEYALgBH
AGUAVABWAEEABAB1AGUAKAAkAE4AVQBzAGWAKQA7AEKARGAoACQARwBQAEMAWwAnAFMAYwByAGKAcAB0
```

```
Windows PowerShell

PS D:\> .\volatility_2.6_win64_standalone.exe -f .\incident.mem --profile=win7SP1x86
cmdline -p 3008,3672
Volatility Foundation Volatility Framework 2.6
*****
cmd.exe pid: 3008
Command line : C:\windows\system32\cmd.exe /c C:\Users\wilfred\AppData\Roaming\Identi
ties\Updater.bat
*****
powershell.exe pid: 3672
Command line :
PS D:\>
```



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:

```

PS D:\> .\volatility_2.6_win64_standalone.exe -f .\nwe.mem --profile=win7/SP1x64 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\mscorlib.wd.dll
1744     nwe.exe      0x000000000012c0000 True   False True   Users\lesley (win 7)\Desktop\nwe.exe
1744     nwe.exe      0x00000000071f0000 False  False False  \Windows\SysWOW64\RPCRT4.dll
1744     nwe.exe      0x000000000073d90000 False  False False  \Windows\SysWOW64\msasn32.dll
1744     nwe.exe      0x00000000073d90000 True   True   True   \Windows\System32\wow64cpu.dll
1744     nwe.exe      0x00000000073d90000 False  False False  \Windows\SysWOW64\urlmon.dll
1744     nwe.exe      0x000000006e430000 False  False False  \Windows\assembly\NativeImages_v2.0.50727_32\mscorlib\62a
0b3e4b40e0c8c5cfaa0c8848e64a\mscorlib.ni.dll
1744     nwe.exe      0x000000000070850000 False  False False  \Windows\winsxs\x86_microsoft.vc80.crt_1fc8b3b9a1e83b_8
.0.50727.4940_none_d08cc06a442b34f4\msvcr80.dll
1744     nwe.exe      0x00000000004b60000 False  False False  \Windows\assembly\GAC_MSIL\Microsoft.VisualBasic.resources
s\8.0.0.0_ru_b03f5f7f11d50a3a\Microsoft.VisualBasic.resources.dll
1744     nwe.exe      0x00000000077440000 False  False False  \Windows\SysWOW64\crypt32.dll
1744     nwe.exe      0x00000000075aa0000 False  False False  \Windows\SysWOW64\setupapi.dll
1744     nwe.exe      0x00000000071990000 False  False False  \Windows\SysWOW64\sxs.dll
1744     nwe.exe      0x00000000070660000 False  False False  \Windows\assembly\NativeImages_v2.0.50727_32\System.Drawi
ng\df8e642a8ed7b2b103ad28e0c96418a\System.Drawing.ni.dll
1744     nwe.exe      0x0000000005e3a0000 False  False False  \Windows\Microsoft.NET\Framework\v2.0.50727\diasymreader.
dll
1744     nwe.exe      0x000000000705b0000 False  False False  \Windows\assembly\GAC_MSIL\Microsoft.VisualBasic.resources
s\8.0.0.0_ru_b03f5f7f11d50a3a\Microsoft.VisualBasic.resources.dll
1744     nwe.exe      0x0000000006fb0000 False  False False  \Windows\SysWOW64\scrnrun.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 that 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 0x00000000003c70000
rundll32.exe 988 0x00000000002980000
powershell.exe 3876 0x00000000005a40000
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 .



in 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 Doppelganging

- 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 doppelganged 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 Doppelganging.



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

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

```

Windows PowerShell
PS D:\> .\volatility_2.6_win64_standalone.exe -f .\Inside.vmem --profile=win10x64_14393 volshell
Volatility Foundation Volatility Framework 2.6
Current context: System @ 0xfffffe00142226040, pid=4, ppid=0 DTB=0x1aa000
Welcome to volshell! Current memory image is:
file:///D:/Inside.vmem
To get help, type 'hh()'
>>> ps()

```

Name	PID	PPID	Offset
System	4	0	0xfffffe00142226040
smss.exe	308	4	0xfffffe001441f9440
csrss.exe	408	396	0xfffffe0014476b080
smss.exe	480	308	0xfffffe00144ddb080
wininit.exe	488	396	0xfffffe00144ddf080
csrss.exe	496	480	0xfffffe00144772840
winlogon.exe	568	480	0xfffffe00144e37080
services.exe	624	488	0xfffffe00144e683c0
lsass.exe	636	488	0xfffffe00144e9d080
svchost.exe	728	624	0xfffffe00144e64840
svchost.exe	784	624	0xfffffe00144216400
svchost.exe	916	624	0xfffffe00144f3c840

```

Select Windows PowerShell
0x430 : DeviceMap 18446673705251523808
0x438 : EtwDataSource 18446708894810156481
0x440 : PageDirectoryPte 0
0x448 : ImageFilePointer 18446708894815916176
0x450 : ImageFileName explorer.exe
0x45f : PriorityClass 2
0x460 : SecurityPort 0
0x468 : SeAuditProcessCreationInfo 18446708894760519080
0x470 : JobLinks 18446708894760519088
0x480 : HighestUserAddress 140737488289792

```

## Looking for evidence of persistence

### 1- Boot or Logon Autostart Execution technique

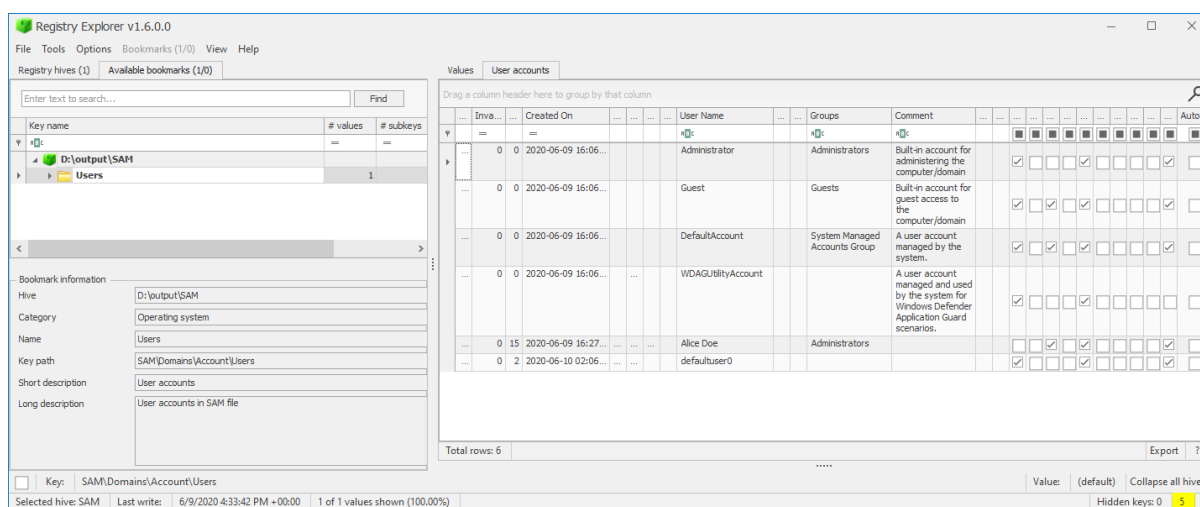
- **HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon**
- **HKLM\Software\Microsoft\Windows\CurrentVersion\Run**
- **HKLM\Software\Microsoft\Windows\CurrentVersion\RunOnce**
- **HKCU\Software\Microsoft\Windows\CurrentVersion\Run**
- **HKCU\Software\Microsoft\Windows\CurrentVersion\RunOnce**
- **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**)



- 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. **svcs** 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.