14th Int. Conf. on IT Security Incident Management & IT Forensics (IMF 2025)
Practical Malware Analysis and Memory Forensics for Incident Response
Albstadt-Sigmaringen University (Germany)

Getting Started with Volatility

In this lab, you will be introduced to malware forensics using Volatility. To complete the exercises, you will use the containerized environment provided by the instructor, along with one of the memory dumps. All additional materials are available on the training workshop website: `https://webdiis.unizar.es/~ricardo/imf-25-workshop`. As an example, you will analyze a memory dump to understand how Volatility works, exploring several of its built-in plugins and interpreting their results.

1 `zeus.vmem` Dump Analysis

As an example of how to use Volatility (specifically, version 2.6.1), we will analyze a memory dump from a Windows computer infected with the notorious ZeuS banking Trojan. This dump is available on the workshop website (`https://webdiis.unizar.es/~ricardo/imf-25-workshop`) and comes from the "Malware Analyst Cookbook"[1]. You can download the compressed memory dump to your host machine and extract it into the folder shared with your container. In the following examples, we assume that this shared folder is mounted as "/shared" inside the container.

1.1 Identifying the Source Machine

Volatility 2.6 needs to know the operating system of the dump being analyzed in order to determine the location of the kernel structures of interest and analyze it accurately. The plugin for this is called `imageinfo` (in case you don't know beforehand the origin). In this example, we can run the following command in the Docker container:

```
vol2 -f /shared/zeus.vmem imageinfo
```

Example output:

```
1 Volatility Foundation Volatility Framework 2.6.1
2 INFO    : volatility.debug    : Determining profile based on KDBG
↪    search...
3           Suggested Profile(s) : WinXPSP2x86, WinXPSP3x86 (
```

[1] "Malware Analyst's Cookbook and DVD: Tools and Techniques for Fighting Malicious Code". Michael Ligh, Steven Adair, Blake Hartstein, and Matthew Richard. Wiley, 1st Ed. (November 2, 2010). ISBN: 978-0470613030.

```
                Instantiated with WinXPSP2x86)
                        AS Layer1 : IA32PagedMemoryPae (Kernel AS)
                        AS Layer2 : FileAddressSpace (/shared/zeus.vmem)
                         PAE type : PAE
                              DTB : 0x319000L
                             KDBG : 0x80544ce0L
          Number of Processors : 1
     Image Type (Service Pack) : 2
                KPCR for CPU 0 : 0xffdff000L
             KUSER_SHARED_DATA : 0xffdf0000L
           Image date and time : 2010-08-15 19:17:56 UTC+0000
     Image local date and time : 2010-08-15 15:17:56 -0400
```

As shown, the recommended profiles are `WinXPSP2x86` and `WinXPSP3x86`.

1.2 Running Processes

The Volatility plugin that allows us to know which processes were running in the analyzed dump is `pslist`. This plugin traverses the doubly linked list pointed to by `PsActiveProcessHead` (internal structure of the Windows kernel) and displays for each process its offset, name, ID (PID), parent process ID (PPID), number of threads, number of handles, session ID, whether it is a WoW64 process, and the start and/or end date and time of execution. WoW64 stands for "Windows-on-Windows 64-bit" and is a subsystem in 64-bit versions of Windows that allows 32-bit applications to run on 64-bit Windows operating systems. By analyzing memory dumps, WoW64 process identification helps determine whether 32-bit code is running in a 64-bit environment, which is important for identifying potential evasion or injection techniques.

Processes that show 0 threads, 0 handles, or an end date and time are inactive processes. Some processes may not be linked to any session. This is because they are created before the session manager.

```
➜  ~ vol2 -f /shared/zeus.vmem --profile=WinXPSP2x86 pslist
Volatility Foundation Volatility Framework 2.6.1
Offset(V)  Name                    PID   PPID   Thds     Hnds   Sess  Wow64 Start                          Exit
---------- -------------------- ------ ------ ------ -------- ------ ------ ------------------------------ ------------------------------
0x810b1660 System                    4      0     58      379 ------      0
0xff2ab020 smss.exe                544      4      3       21 ------      0 2010-08-11 06:06:21 UTC+0000
0xff1ecda0 csrss.exe               608    544     10      410      0      0 2010-08-11 06:06:23 UTC+0000
0xff1ec978 winlogon.exe            632    544     24      536      0      0 2010-08-11 06:06:23 UTC+0000
0xff247020 services.exe            676    632     16      288      0      0 2010-08-11 06:06:24 UTC+0000
0xff255020 lsass.exe               688    632     21      405      0      0 2010-08-11 06:06:24 UTC+0000
0xff218230 vmacthlp.exe            844    676      1       37      0      0 2010-08-11 06:06:24 UTC+0000
0x80ff88d8 svchost.exe             856    676     29      336      0      0 2010-08-11 06:06:24 UTC+0000
0xff217560 svchost.exe             936    676     11      288      0      0 2010-08-11 06:06:24 UTC+0000
0x80fbf910 svchost.exe            1028    676     88     1424      0      0 2010-08-11 06:06:24 UTC+0000
0xff22d558 svchost.exe            1088    676      7       93      0      0 2010-08-11 06:06:25 UTC+0000
0xff203b80 svchost.exe            1148    676     15      217      0      0 2010-08-11 06:06:26 UTC+0000
0xff1d7da0 spoolsv.exe            1432    676     14      145      0      0 2010-08-11 06:06:26 UTC+0000
0xff1b8b28 vmtoolsd.exe           1668    676      5      225      0      0 2010-08-11 06:06:35 UTC+0000
0xff1fdc88 VMUpgradeHelper        1788    676      5      112      0      0 2010-08-11 06:06:38 UTC+0000
0xff143b28 TPAutoConnSvc.e        1968    676      5      106      0      0 2010-08-11 06:06:39 UTC+0000
0xff25a7e0 alg.exe                 216    676      8      120      0      0 2010-08-11 06:06:39 UTC+0000
0xff364310 wscntfy.exe             888   1028      1       40      0      0 2010-08-11 06:06:49 UTC+0000
0xff38b5f8 TPAutoConnect.e        1084   1968      1       68      0      0 2010-08-11 06:06:52 UTC+0000
0x80f60da0 wuauclt.exe            1732   1028      7      189      0      0 2010-08-11 06:07:44 UTC+0000
0xff3865d0 explorer.exe           1724   1708     13      326      0      0 2010-08-11 06:09:29 UTC+0000
0xff3667e8 VMwareTray.exe          432   1724      1       60      0      0 2010-08-11 06:09:31 UTC+0000
0xff374980 VMwareUser.exe          452   1724      8      207      0      0 2010-08-11 06:09:32 UTC+0000
0x80f94588 wuauclt.exe             468   1028      4      142      0      0 2010-08-11 06:09:37 UTC+0000
0xff224020 cmd.exe                 124   1668      0 --------      0      0 2010-08-15 19:17:55 UTC+0000   2010-08-15 19:17:56 UTC+0000
```


Hidden or unlinked processes from the active process list will not appear in `pslist`. In these cases, use the `psscan` plugin, which searches for process objects in memory without relying on standard lists.

```
➜  ~ vol2 -f /shared/zeus.vmem --profile=WinXPSP2x86 psscan
Volatility Foundation Volatility Framework 2.6.1
Offset(P)          Name                PID   PPID PDB        Time created                   Time exited
------------------ ---------------- ------ ------ ---------- ------------------------------ ------------------------------
0x00000000010c3da0 wuauclt.exe        1732   1028 0x06cc02c0 2010-08-11 06:07:44 UTC+0000
0x00000000010f7588 wuauclt.exe         468   1028 0x06cc0180 2010-08-11 06:09:37 UTC+0000
0x0000000001122910 svchost.exe        1028    676 0x06cc0120 2010-08-11 06:06:24 UTC+0000
0x000000000115b8d8 svchost.exe         856    676 0x06cc00e0 2010-08-11 06:06:24 UTC+0000
0x0000000001214660 System                4      0 0x00319000
0x000000000211ab28 TPAutoConnSvc.e    1968    676 0x06cc0260 2010-08-11 06:06:39 UTC+0000
0x00000000049c15f8 TPAutoConnect.e    1084   1968 0x06cc0220 2010-08-11 06:06:52 UTC+0000
0x0000000004a065d0 explorer.exe       1724   1708 0x06cc0280 2010-08-11 06:09:29 UTC+0000
0x0000000004b5a980 VMwareUser.exe      452   1724 0x06cc0300 2010-08-11 06:09:32 UTC+0000
0x0000000004be97e8 VMwareTray.exe      432   1724 0x06cc02e0 2010-08-11 06:09:31 UTC+0000
0x0000000004c2b310 wscntfy.exe         888   1028 0x06cc0200 2010-08-11 06:06:49 UTC+0000
0x0000000005471020 smss.exe            544      4 0x06cc0020 2010-08-11 06:06:21 UTC+0000
0x0000000005f027e0 alg.exe             216    676 0x06cc0240 2010-08-11 06:06:39 UTC+0000
0x0000000005f47020 lsass.exe           688    632 0x06cc00a0 2010-08-11 06:06:24 UTC+0000
0x0000000006015020 services.exe        676    632 0x06cc0080 2010-08-11 06:06:24 UTC+0000
0x00000000061ef558 svchost.exe        1088    676 0x06cc0140 2010-08-11 06:06:25 UTC+0000
0x0000000006238020 cmd.exe             124   1668 0x06cc02a0 2010-08-15 19:17:55 UTC+0000   2010-08-15 19:17:56 UTC+0000
0x0000000006384230 vmacthlp.exe        844    676 0x06cc00c0 2010-08-11 06:06:24 UTC+0000
0x00000000063c5560 svchost.exe         936    676 0x06cc0100 2010-08-11 06:06:24 UTC+0000
0x0000000006499b80 svchost.exe        1148    676 0x06cc0160 2010-08-11 06:06:26 UTC+0000
0x000000000655fc88 VMUpgradeHelper    1788    676 0x06cc01e0 2010-08-11 06:06:38 UTC+0000
0x00000000066f0978 winlogon.exe        632    544 0x06cc0060 2010-08-11 06:06:23 UTC+0000
0x00000000066f0da0 csrss.exe           608    544 0x06cc0040 2010-08-11 06:06:23 UTC+0000
0x0000000006945da0 spoolsv.exe        1432    676 0x06cc01a0 2010-08-11 06:06:26 UTC+0000
0x00000000069a7328 VMip.exe           1944    124 0x06cc0320 2010-08-15 19:17:55 UTC+0000   2010-08-15 19:17:56 UTC+0000
0x00000000069d5b28 vmtoolsd.exe       1668    676 0x06cc01c0 2010-08-11 06:06:35 UTC+0000
```

To visualize process hierarchies, use the `pstree` plugin, which displays parent-child relationships in a tree structure.

```
➜  ~ vol2 -f /shared/zeus.vmem --profile=WinXPSP2x86 pstree
Volatility Foundation Volatility Framework 2.6.1
Name                                                  Pid   PPid   Thds   Hnds Time
-------------------------------------------------- ------ ------ ------ ------ ----
 0x810b1660:System                                      4      0     58    379 1970-01-01 00:00:00 UTC+0000
. 0xff2ab020:smss.exe                                 544      4      3     21 2010-08-11 06:06:21 UTC+0000
.. 0xff1ec978:winlogon.exe                            632    544     24    536 2010-08-11 06:06:23 UTC+0000
... 0xff255020:lsass.exe                              688    632     21    405 2010-08-11 06:06:24 UTC+0000
... 0xff247020:services.exe                           676    632     16    288 2010-08-11 06:06:24 UTC+0000
.... 0xff1b8b28:vmtoolsd.exe                         1668    676      5    225 2010-08-11 06:06:35 UTC+0000
..... 0xff224020:cmd.exe                              124   1668      0 ------ 2010-08-15 19:17:55 UTC+0000
.... 0x80ff88d8:svchost.exe                           856    676     29    336 2010-08-11 06:06:24 UTC+0000
.... 0xff1d7da0:spoolsv.exe                          1432    676     14    145 2010-08-11 06:06:26 UTC+0000
.... 0x80fbf910:svchost.exe                          1028    676     88   1424 2010-08-11 06:06:24 UTC+0000
..... 0x80f60da0:wuauclt.exe                         1732   1028      7    189 2010-08-11 06:07:44 UTC+0000
..... 0x80f94588:wuauclt.exe                          468   1028      4    142 2010-08-11 06:09:37 UTC+0000
..... 0xff364310:wscntfy.exe                          888   1028      1     40 2010-08-11 06:06:49 UTC+0000
.... 0xff217560:svchost.exe                           936    676     11    288 2010-08-11 06:06:24 UTC+0000
.... 0xff143b28:TPAutoConnSvc.e                      1968    676      5    106 2010-08-11 06:06:39 UTC+0000
..... 0xff38b5f8:TPAutoConnect.e                     1084   1968      1     68 2010-08-11 06:06:52 UTC+0000
.... 0xff22d558:svchost.exe                          1088    676      7     93 2010-08-11 06:06:25 UTC+0000
.... 0xff218230:vmacthlp.exe                          844    676      1     37 2010-08-11 06:06:24 UTC+0000
.... 0xff25a7e0:alg.exe                               216    676      8    120 2010-08-11 06:06:39 UTC+0000
.... 0xff203b80:svchost.exe                          1148    676     15    217 2010-08-11 06:06:26 UTC+0000
.... 0xff1fdc88:VMUpgradeHelper                      1788    676      5    112 2010-08-11 06:06:38 UTC+0000
.. 0xff1ecda0:csrss.exe                               608    544     10    410 2010-08-11 06:06:23 UTC+0000
 0xff3865d0:explorer.exe                             1724   1708     13    326 2010-08-11 06:09:29 UTC+0000
. 0xff374980:VMwareUser.exe                           452   1724      8    207 2010-08-11 06:09:32 UTC+0000
. 0xff3667e8:VMwareTray.exe                           432   1724      1     60 2010-08-11 06:09:31 UTC+0000
```


The `psxview` plugin is another valuable tool for detecting hidden processes. It compares the results of multiple enumeration techniques (e.g., `PsActiveProcessHead`, `EPROCESS`, and `ETHREAD` objects, `PspCidTable`, and `csrss.exe` references).

```
➜  ~ vol2 -f /shared/zeus.vmem --profile=WinXPSP2x86 psxview
Volatility Foundation Volatility Framework 2.6.1
Offset(P)  Name                    PID pslist psscan thrdproc pspcid csrss session deskthrd ExitTime
---------- -------------------- ------ ------ ------ -------- ------ ----- ------- -------- --------
0x06015020 services.exe            676 True   True   True     True   True  True    True
0x063c5560 svchost.exe             936 True   True   True     True   True  True    True
0x06499b80 svchost.exe            1148 True   True   True     True   True  True    True
0x04c2b310 wscntfy.exe             888 True   True   True     True   True  True    True
0x049c15f8 TPAutoConnect.e        1084 True   True   True     True   True  True    True
0x05f027e0 alg.exe                 216 True   True   True     True   True  True    True
0x05f47020 lsass.exe               688 True   True   True     True   True  True    True
0x010f7588 wuauclt.exe             468 True   True   True     True   True  True    True
0x01122910 svchost.exe            1028 True   True   True     True   True  True    True
0x069d5b28 vmtoolsd.exe           1668 True   True   True     True   True  True    True
0x06384230 vmacthlp.exe            844 True   True   True     True   True  True    True
0x0115b8d8 svchost.exe             856 True   True   True     True   True  True    True
0x04b5a980 VMwareUser.exe          452 True   True   True     True   True  True    True
0x010c3da0 wuauclt.exe            1732 True   True   True     True   True  True    True
0x04a065d0 explorer.exe           1724 True   True   True     True   True  True    True
0x04be97e8 VMwareTray.exe          432 True   True   True     True   True  True    True
0x0211ab28 TPAutoConnSvc.e        1968 True   True   True     True   True  True    True
0x06945da0 spoolsv.exe            1432 True   True   True     True   True  True    True
0x066f0978 winlogon.exe            632 True   True   True     True   True  True    True
0x0655fc88 VMUpgradeHelper        1788 True   True   True     True   True  True    True
0x061ef558 svchost.exe            1088 True   True   True     True   True  True    True
0x06238020 cmd.exe                 124 True   True   False    True   False False   False    2010-08-15 19:17:56 UTC+0000
0x066f0da0 csrss.exe               608 True   True   True     True   False True    True
0x05471020 smss.exe                544 True   True   True     True   False False   False
0x01214660 System                    4 True   True   True     True   False False   False
0x069a7328 VMip.exe               1944 False  True   False    False  False False   False    2010-08-15 19:17:56 UTC+0000
```

For process analysis, the `dlllist` plugin displays the dynamic-link libraries (DLLs) loaded by each process. You can limit the output to a specific process using the `-p` parameter.


```
➜  ~ vol2 -f /shared/zeus.vmem --profile=WinXPSP2x86 dlllist | more
Volatility Foundation Volatility Framework 2.6.1
************************************************************************
System pid:      4
Unable to read PEB for task.
************************************************************************
smss.exe pid:    544
Command line : \SystemRoot\System32\smss.exe


Base             Size  LoadCount LoadTime                       Path
---------- ---------- ---------- ------------------------------ ----
0x48580000     0xf000     0xffff                                \SystemRoot\System32\smss.exe
0x7c900000    0xb0000     0xffff
************************************************************************
csrss.exe pid:    608
Command line : C:\WINDOWS\system32\csrss.exe ObjectDirectory=\Windows SharedSection=1024,3072,512 Windows=On
SubSystemType=Windows ServerDll=basesrv,1 Se
rverDll=winsrv:UserServerDllInitialization,3 ServerDll=winsrv:ConServerDllInitialization,2 ProfileControl=Off
 MaxRequestThreads=16
Service Pack 2

Base             Size  LoadCount LoadTime                       Path
---------- ---------- ---------- ------------------------------ ----
0x4a680000     0x5000     0xffff                                \??\C:\WINDOWS\system32\csrss.exe
0x7c900000    0xb0000     0xffff                                C:\WINDOWS\system32\ntdll.dll
0x75b40000     0xb000     0xffff                                C:\WINDOWS\system32\CSRSRV.dll
0x75b50000    0x10000        0x3                                C:\WINDOWS\system32\basesrv.dll
0x75b60000     0x4a000        0x2                                C:\WINDOWS\system32\winsrv.dll
0x77d40000    0x90000        0x6                                C:\WINDOWS\system32\USER32.dll
0x7c800000    0xf4000        0xe                                C:\WINDOWS\system32\KERNEL32.dll
0x77f10000    0x46000        0x5                                C:\WINDOWS\system32\GDI32.dll
0x75e90000    0xb0000        0x1                                C:\WINDOWS\system32\sxs.dll
0x77dd0000    0x9b000        0x3                                C:\WINDOWS\system32\ADVAPI32.dll
```

To inspect the kernel drivers loaded at the time of the dump, use the `modscan` plugin.

```
➜  ~ vol2 -f /shared/zeus.vmem --profile=WinXPSP2x86 modscan
Volatility Foundation Volatility Framework 2.6.1
Offset(P)          Name                Base             Size File
------------------ ------------------- ---------- ---------- ----
0x0000000001058d80 serenum.sys         0xfc93b000     0x4000 \SystemRoot\system32\DRIVERS\serenum.sys
0x000000000105ad70 vmmemctl.sys        0xfc9f7000     0x2000 \??\C:\Program Files\VMware\VMware Tools\Driver
s\memctl\vmmemctl.sys
0x000000000105f0c8 dump_vmscsi.sys     0xfbf36000     0x3000 \SystemRoot\System32\Drivers\dump_vmscsi.sys
0x00000000010664a8 srv.sys             0xf355d000    0x53000 \SystemRoot\system32\DRIVERS\srv.sys
0x0000000001067700 mrxdav.sys          0xf35d8000    0x2d000 \SystemRoot\system32\DRIVERS\mrxdav.sys
0x000000000106f050 rasacd.sys          0xfc174000     0x3000 \SystemRoot\system32\DRIVERS\rasacd.sys
0x000000000106f8c8 Msfs.SYS            0xfc7c3000     0x5000 \SystemRoot\System32\Drivers\Msfs.SYS
0x0000000001070e60 i8042prt.sys        0xfc53b000     0xd000 \SystemRoot\system32\DRIVERS\i8042prt.sys
0x000000000108be28 dump_scsiport.sys   0xfbf3a000     0x4000 \SystemRoot\System32\Drivers\dump_diskdump.sys
0x000000000108c008 watchdog.sys        0xfc7f3000     0x5000 \SystemRoot\System32\watchdog.sys
0x000000000108f340 HIDPARSE.SYS        0xfc7eb000     0x7000 \SystemRoot\system32\DRIVERS\HIDPARSE.SYS
0x0000000001092008 mouhid.sys          0xfbf3e000     0x3000 \SystemRoot\system32\DRIVERS\mouhid.sys
0x0000000001093690 ndiswan.sys         0xfc08c000    0x17000 \SystemRoot\system32\DRIVERS\ndiswan.sys
0x0000000001093b18 rasl2tp.sys         0xfc5cb000     0xd000 \SystemRoot\system32\DRIVERS\rasl2tp.sys
0x0000000001093ec8 ptilink.sys         0xfc79b000     0x5000 \SystemRoot\system32\DRIVERS\ptilink.sys
0x00000000010ad638 raspti.sys          0xfc7a3000     0x5000 \SystemRoot\system32\DRIVERS\raspti.sys
0x00000000010afe78 update.sys          0xfbf4e000    0x34000 \SystemRoot\system32\DRIVERS\update.sys
0x00000000010b06d8 swenum.sys          0xfc9a5000     0x2000 \SystemRoot\system32\DRIVERS\swenum.sys
0x00000000010b20c0 HIDCLASS.SYS        0xfc6fb000     0x9000 \SystemRoot\system32\DRIVERS\HIDCLASS.SYS
0x00000000010b8508 HTTP.sys            0xf329c000    0x41000 \SystemRoot\System32\Drivers\HTTP.sys
0x00000000010c7800 ParVdm.SYS          0xfc9f5000     0x2000 \SystemRoot\System32\Drivers\ParVdm.SYS
```


Finally, the `threads` plugin provides detailed information about each thread, including the state of CPU registers, entry point disassembly, and other relevant fields for further investigation.

```
➜  ~ vol2 -f /shared/zeus.vmem --profile=WinXPSP2x86 threads | more
Volatility Foundation Volatility Framework 2.6.1
[x86] Gathering all referenced SSDTs from KTHREADs...
Finding appropriate address space for tables...
------
ETHREAD: 0xff242800 Pid: 1028 Tid: 1564
Tags:
Created: 2010-08-11 06:06:35 UTC+0000
Exited: 1970-01-01 00:00:00 UTC+0000
Owning Process: svchost.exe
Attached Process: svchost.exe
State: Waiting:WrLpcReceive
BasePriority: 0x8
Priority: 0x8
TEB: 0x7ff9c000
StartAddress: 0x7c810856 kernel32.dll
ServiceTable: 0x80552180
  [0] 0x80501030
  [1] 0x00000000
  [2] 0x00000000
  [3] 0x00000000
Win32Thread: 0x00000000
CrossThreadFlags:
Eip: 0x7c90eb94
  eax=0x77e76bf0 ebx=0x00000000 ecx=0x00bafb3e edx=0x000e000c esi=0x000d16e8 edi=0x000d178c
  eip=0x7c90eb94 esp=0x013cfe1c ebp=0x013cff80 err=0x00000000
  cs=0x1b ss=0x23 ds=0x23 es=0x23 gs=0x00 fs=0x3b efl=0x00000246
  dr0=0x00000000 dr1=0x00000000 dr2=0x00000000 dr3=0x00000000 dr6=0x00000000 dr7=0x00000000
0x7c810856 33ed            XOR EBP, EBP
0x7c810858 53              PUSH EBX
0x7c810859 50              PUSH EAX
0x7c81085a 6a00            PUSH 0x0
0x7c81085c e973acffff      JMP 0x7c80b4d4
0x7c810861 90              NOP
```

By default, this plugin lists all threads on the system. You can filter the output using the `-F` (or `--filter`) parameter. Multiple filters can be specified, separated by commas. To see the available filters, use the `-L` option:

```
➜  ~ vol2 -f /shared/zeus.vmem --profile=WinXPSP2x86 threads -L
Volatility Foundation Volatility Framework 2.6.1
Impersonation        Detect impersonating threads
DkomExit             Detect inconsistencies wrt exit times and termination
ScannerOnly          Detect threads no longer in a linked list
SystemThread         Detect system threads
HideFromDebug        Detect threads hidden from debuggers
OrphanThread         Detect orphan threads
AttachedProcess      Detect threads attached to another process
HookedSSDT           Check if a thread is using a hooked SSDT
HwBreakpoint         Detect threads with hardware breakpoints
```


1.3 Internet Connection

The `connscan` plugin searches for `_TCPT_OBJECT` structures in memory, which represent network connection objects, both those that were active at the time of the memory acquisition and those that were recently closed. For each object found, it displays the memory address, local and remote IP addresses, and the PID associated with the connection. However, note that the PID is not always retrieved accurately, which can lead to false positives. This plugin is only supported on Windows XP and Windows Server 2003.

```
➜  ~ vol2 -f /shared/zeus.vmem --profile=WinXPSP2x86 connscan
Volatility Foundation Volatility Framework 2.6.1
Offset(P)  Local Address             Remote Address            Pid
---------- ------------------------ ------------------------ ---
0x02214988 172.16.176.143:1054      193.104.41.75:80         856
0x06015ab0 0.0.0.0:1056             193.104.41.75:80         856
```

In the example above, two connections related to the process with PID 856 are displayed. As shown earlier in Section 1.2, this PID corresponds to the `svchost.exe` process, a generic host process for services running from DLLs. Its parent process is `services.exe` (PID 676), which is responsible for managing system services and starting `svchost.exe` instances during system boot.

To display only active connections at the time of the memory acquisition, use the `connections` plugin.

Another relevant plugin is `sockets`, which lists all open sockets of any type (TCP, UDP, RAW, etc.). However, please note that this plugin may not work correctly on modern versions of Windows. Like `connscan`, it is limited to Windows XP and Windows Server 2003.

```
➜  ~ vol2 -f /shared/zeus.vmem --profile=WinXPSP2x86 sockets
Volatility Foundation Volatility Framework 2.6.1
Offset(V)       PID   Port  Proto Protocol        Address         Create Time
---------- -------- ------ ------ --------------- --------------- -----------
0x80fd1008        4      0     47 GRE             0.0.0.0         2010-08-11 06:08:00 UTC+0000
0xff258008      688    500     17 UDP             0.0.0.0         2010-08-11 06:06:35 UTC+0000
0xff367008        4    445      6 TCP             0.0.0.0         2010-08-11 06:06:17 UTC+0000
0x80ffc128      936    135      6 TCP             0.0.0.0         2010-08-11 06:06:24 UTC+0000
0xff37cd28     1028   1058      6 TCP             0.0.0.0         2010-08-15 19:17:56 UTC+0000
0xff20c478      856  29220      6 TCP             0.0.0.0         2010-08-15 19:17:27 UTC+0000
0xff225b70      688      0    255 Reserved        0.0.0.0         2010-08-11 06:06:35 UTC+0000
0xff254008     1028    123     17 UDP             127.0.0.1       2010-08-15 19:17:56 UTC+0000
0x80fce930     1088   1025     17 UDP             0.0.0.0         2010-08-11 06:06:38 UTC+0000
0xff127d28      216   1026      6 TCP             127.0.0.1       2010-08-11 06:06:39 UTC+0000
0xff206a20     1148   1900     17 UDP             127.0.0.1       2010-08-15 19:17:56 UTC+0000
0xff1b8250      688   4500     17 UDP             0.0.0.0         2010-08-11 06:06:35 UTC+0000
0xff382e98        4   1033      6 TCP             0.0.0.0         2010-08-11 06:08:00 UTC+0000
0x80fbdc40        4    445     17 UDP             0.0.0.0         2010-08-11 06:06:17 UTC+0000
```

For newer versions of Windows (e.g., Windows 7 and later), use the `netscan` plugin. We'll demonstrate its use in a later laboratory.

1.4 File Handles and Artifacts

To examine files and other resources associated with specific processes, you can use the `handles` plugin. This plugin supports the `-t` parameter, which allows you to filter by handle type.

For example, to inspect mutex objects associated with the previously identified process (PID 856), we can specify the `Mutant` type (Windows refers to mutexes as "mutants"):

```
➜  ~ vol2 -f /shared/zeus.vmem --profile=WinXPSP2x86 handles -t Mutant -p 856
Volatility Foundation Volatility Framework 2.6.1
Offset(V)     Pid     Handle     Access Type            Details
---------- ------ ---------- ---------- ---------------- -------
0xff257148    856       0x24   0x1f0001 Mutant           SHIMLIB_LOG_MUTEX
0xff149878    856      0x158   0x1f0001 Mutant
0xff2342e8    856      0x1d8   0x1f0001 Mutant
0xff3864f8    856      0x1e4   0x120001 Mutant           ShimCacheMutex
0xff21e0e0    856      0x1ec   0x1f0001 Mutant
0xff22f0e0    856      0x1f8   0x1f0001 Mutant
0xff2232e8    856      0x200   0x1f0001 Mutant
0xff2741f0    856      0x218   0x1f0001 Mutant           746bbf3569adEncrypt
0xff15a2c0    856      0x238   0x1f0001 Mutant
0x80fca0e0    856      0x288   0x1f0001 Mutant
0x80ef7a38    856      0x3d4   0x100000 Mutant           _!MSFTHISTORY!_
0x80fdc1b8    856      0x3dc   0x1f0001 Mutant           c:!windows!system32!config!systemprofile!local setti
ngs!temporary internet files!content.ie5!
0x80f18290    856      0x3e0   0x1f0001 Mutant           c:!windows!system32!config!systemprofile!cookies!
0x80fbbb40    856      0x3ec   0x1f0001 Mutant           c:!windows!system32!config!systemprofile!local setti
ngs!history!history.ie5!
0x80fbe1a8    856      0x3f8   0x1f0001 Mutant           ZonesCacheCounterMutex
0x80f66898    856      0x3fc   0x1f0001 Mutant           ZonesCounterMutex
0x80f30c90    856      0x404   0x1f0001 Mutant           ZonesLockedCacheCounterMutex
0xff2071d0    856      0x418   0x100000 Mutant           WininetStartupMutex
0xff1e3d48    856      0x420   0x1f0001 Mutant
0x80f27f60    856      0x424   0x1f0001 Mutant
0x80f0cb60    856      0x428   0x100000 Mutant           WininetProxyRegistryMutex
0xff27b7e8    856      0x43c   0x1f0001 Mutant           _AVIRA_2108
0x80f19200    856      0x450   0x1f0001 Mutant
0xff1e68b0    856      0x460   0x100000 Mutant           RasPbFile
```

Here you can see a mutex named `_AVIRA_2108`. This mutex name is known to be associated with the ZeuS banking Trojan, which uses it as a marker to indicate that malware is already running on the system. Mutexes (short for *mutual exclusion* objects) are synchronization primitives used by processes to prevent multiple instances from running simultaneously or to coordinate access to shared resources. In the case of ZeuS, creating a specific mutex, such as `_AVIRA_XXX`, serves to prevent re-infection by another instance of the same malware.

This is a common technique among malware families to ensure that only one copy of the malware runs at a time. If a new copy runs and detects that the mutex already exists, it will typically delete itself to prevent unnecessary duplication or potential detection.

Early variants of ZeuS typically used the `winlogon.exe` process as a persistence host. `winlogon.exe` is a critical system process responsible for managing user logons, secure attention sequences (such as Ctrl+Alt+Del.), and other session-related tasks. By injecting malicious code into this process or executing under its identity, ZeuS could remain memory-resident and operate with elevated privileges, making it difficult to detect and remove.

This use of known system processes and subtle artifacts like mutexes is a hallmark of the malware's stealthy behavior. It also provides forensic analysts with valuable indicators of compromise (IoCs) when analyzing memory dumps or live systems.

We can also inspect the mutexes associated with the `winlogon.exe` process (PID 632) using the `-t Mutant` parameter and filter the results with `grep`:

```
vol2 -f /shared/zeus.vmem --profile=WinXPSP2x86 handles -t Mutant | grep 632
```


As shown, the same mutex name, `_AVIRA_`, also appears here (but with a different ending number).

We can now search for file identifiers related to `winlogon.exe` by specifying `-t File`. For instance:

```
➜  ~ vol2 -f /shared/zeus.vmem --profile=WinXPSP2x86 handles -t File -p 632
Volatility Foundation Volatility Framework 2.6.1
Offset(V)     Pid     Handle     Access Type            Details
---------- ------ ---------- ---------- --------------- -------
0x81003028    632       0x9c   0x12019f File            \Device\NamedPipe\TerminalServer\AutoReconnect
0xff24b4d0    632       0xd4   0x100020 File            \Device\HarddiskVolume1\WINDOWS\WinSxS\x86_Microsoft.Windows.
0xff257d20    632       0xf4   0x100001 File            \Device\KsecDD
0x80ff7b90    632      0x104   0x120089 File            \Device\HarddiskVolume1\WINDOWS\system32\lowsec\user.ds
0xff224690    632      0x10c   0x12019f File            \Device\NamedPipe\winlogonrpc
0xff23b740    632      0x164   0x100020 File            \Device\HarddiskVolume1\WINDOWS\WinSxS\x86_Microsoft.Windows.
0x81025968    632      0x178   0x12019f File            \Device\NamedPipe\InitShutdown
0x81025598    632      0x17c   0x12019f File            \Device\NamedPipe\InitShutdown
0xff1357f0    632      0x1c4   0x100020 File            \Device\HarddiskVolume1\WINDOWS\system32
0xff21feb8    632      0x204   0x160001 File            \Device\HarddiskVolume1\WINDOWS\AppPatch
0xff220c28    632      0x208   0x160001 File            \Device\HarddiskVolume1\WINDOWS\system32\dllcache
0x80fd0028    632      0x20c   0x160001 File            \Device\HarddiskVolume1\Program Files\Common Files\Microsoft
0xff21ff90    632      0x210   0x160001 File            \Device\HarddiskVolume1\Program Files\Common Files\Microsoft
0xff283550    632      0x214   0x160001 File            \Device\HarddiskVolume1\WINDOWS\system32
0xff21fb68    632      0x218   0x160001 File            \Device\HarddiskVolume1\WINDOWS\Help
0xff27af90    632      0x21c   0x160001 File            \Device\HarddiskVolume1\Program Files\Common Files\Microsoft
0xff12bb40    632      0x644   0x120089 File            \Device\HarddiskVolume1\WINDOWS\system32\sdra64.exe
0xff13a470    632      0x648   0x120089 File            \Device\HarddiskVolume1\WINDOWS\system32\lowsec\local.ds
0xff224768    632      0x6c4   0x12019f File            \Device\NamedPipe\winlogonrpc
0x80f68228    632      0x6d4   0x12019f File            \Device\NamedPipe\lsarpc
0xff1e6b10    632      0x6dc   0x120116 File            \Device\Tcp
0xff1e6a38    632      0x6e0   0x1200a0 File            \Device\Tcp
0xff206778    632      0x6e4   0x1200a0 File            \Device\Ip
0xff1e6610    632      0x6e8   0x100003 File            \Device\Ip
0xff1e6578    632      0x6ec   0x1200a0 File            \Device\Ip
0x80f2c298    632      0x780   0x100020 File            \Device\HarddiskVolume1\WINDOWS\WinSxS\x86_Microsoft.Windows.C
0xff135930    632      0x810   0x12019f File            \Device\KSENUM#00000001\{9B365890-165F-11D0-A195-0020AFD156E4}
0xff3af028    632      0x83c   0x12019f File            \Device\NamedPipe\winlogonrpc
0x80f5cd78    632      0x898   0x12019f File            \Device\NamedPipe\_AVIRA_2109
```

In the output, there are references to files such as `user.ds`, `local.ds`, and the binary `sdra64.exe`. These are all known artifacts associated with the ZeuS banking Trojan. The `user.ds` and `local.ds` files are data storage files used by ZeuS to store stolen information and configuration details. Specifically, `user.ds` typically contains user credentials, such as usernames, passwords, and other sensitive information obtained through techniques such as keylogging or form capture, while `local.ds` typically stores internal configuration data, which may include instructions on how the malware should behave on the infected system, such as command and control server addresses, target domains, and which plugins to load. These files are encrypted or obfuscated to prevent easy detection and analysis, but their presence is a strong indicator of vulnerability.

The `sdra64.exe` binary is particularly significant, as it is the main executable used to deploy and execute the ZeuS malware on the system. It typically copies itself to a system folder, often in the Windows or system32 directory, to integrate with legitimate files. Once installed, it can modify the Windows Registry to achieve persistence, thus ensuring its restart after a system reboot. It can also infiltrate legitimate system processes such as `winlogon.exe`, as mentioned above, to operate stealthily and with elevated privileges.

The presence of `sdra64.exe` alongside `.ds` files in memory is a clear sign of ZeuS infection and provides valuable evidence in forensic investigations.

Finally, notice the presence of a named pipe that matches the mutex. Named pipes are often used for interprocess communication (IPC) or to redirect command output to files on disk.

1.5 Analyzing the Windows Registry

The Windows Registry contains volatile information (dynamically generated during runtime) and static information (loaded from disk). During system startup, a set of registry hive files is loaded to rebuild the registry, which is essential for the proper functioning of the Windows operating system.

To locate the memory addresses and file paths of the different registry hives, you can use the `hivelist` plugin:

```
➜  ~ vol2 -f /shared/zeus.vmem --profile=WinXPSP2x86 hivelist
Volatility Foundation Volatility Framework 2.6.1
Virtual    Physical   Name
---------- ---------- ----
0xe1c49008 0x036dc008 \Device\HarddiskVolume1\Documents and Settings\LocalService\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
0xe1c41b60 0x04010b60 \Device\HarddiskVolume1\Documents and Settings\LocalService\NTUSER.DAT
0xe1a39638 0x021eb638 \Device\HarddiskVolume1\Documents and Settings\NetworkService\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
0xe1a33008 0x01f98008 \Device\HarddiskVolume1\Documents and Settings\NetworkService\NTUSER.DAT
0xe153ab60 0x06b7db60 \Device\HarddiskVolume1\WINDOWS\system32\config\software
0xe1542008 0x06c48008 \Device\HarddiskVolume1\WINDOWS\system32\config\default
0xe1537b60 0x06ae4b60 \SystemRoot\System32\Config\SECURITY
0xe1544008 0x06c4b008 \Device\HarddiskVolume1\WINDOWS\system32\config\SAM
0xe13ae580 0x01bbd580 [no name]
0xe101b008 0x01867008 \Device\HarddiskVolume1\WINDOWS\system32\config\system
0xe1008978 0x01824978 [no name]
0xe1e158c0 0x009728c0 \Device\HarddiskVolume1\Documents and Settings\Administrator\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
0xe1da4008 0x00f6e008 \Device\HarddiskVolume1\Documents and Settings\Administrator\NTUSER.DAT
```

The last column of the output shows the file path on disk and indicates which part of the Windows Registry each file corresponds to. For example, the file `ntuser.dat` contains data associated with the `HKCU` (HKEY_CURRENT_USER) hive, which stores user-specific settings and preferences, such as desktop settings, recently accessed files, and software settings for the currently logged-in user.

These files, also known as registry hives, are critical components of the Windows operating system, and their presence in memory allows Volatility to analyze them even in a purely volatile context. However, if you also have access to the disk image of the compromised system, you can perform a more comprehensive registry forensic analysis. This allows for examining not only the data present in memory at the time of acquisition, but also historical registry keys that were not loaded into memory during the session, registry hives from other user profiles not active at the time of capture, among other metadata.

By combining memory and disk analysis, forensic investigators can piece together a more complete picture of system configuration, usage patterns, malware persistence mechanisms (e.g., executable keys, services, startup folders), and potentially detect manipulation or evasion techniques employed by the attacker.

To list all subkeys of a specific key, you can use the `hivedump` plugin, which provides the virtual address of the target key with the -o parameter:

```
➜  ~ vol2 -f /shared/zeus.vmem --profile=WinXPSP2x86 hivedump -o 0xe1c49008
Volatility Foundation Volatility Framework 2.6.1
Last Written         Key
2010-06-10 16:11:25 UTC+0000 \S-1-5-19_Classes
```

To query a specific registry key, use the `printkey` plugin. Note that the key must be in memory at the time of capture; otherwise, its value cannot be retrieved. In the following instance, we attempt to check if Windows Firewall was enabled by querying the following key: `ControlSet001\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile`.

```
➜  ~ vol2 -f /shared/zeus.vmem --profile=WinXPSP2x86 printkey --key="ControlSet001\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile"
Volatility Foundation Volatility Framework 2.6.1
Legend: (S) = Stable   (V) = Volatile

----------------------------
Registry: \Device\HarddiskVolume1\WINDOWS\system32\config\system
Key name: StandardProfile (S)
Last updated: 2010-08-15 19:17:24 UTC+0000

Subkeys:
  (S) AuthorizedApplications

Values:
REG_DWORD     EnableFirewall  : (S) 0
```

As shown in the output, Windows Firewall was not enabled at the time of the memory capture.

1.6 Memory Artifact Dump

1.6.1 `procdump` Plugin

The `procdump` plugin allows you to dump a process's memory image to a disk file. It supports the `-u` (or `--unsafe`) parameter to bypass certain checks when parsing the portable executable (PE) header. This can be useful, as some malware modifies the PE header to prevent standard tools from extracting it.

Another useful option is `--memory`, which generates a dump based on the process's memory layout, including the additional padding and alignment that the operating system applies when loading the executable into memory.

1.6.2 `dlldump` Plugin

The `dlldump` plugin is used to extract DLLs from memory to disk. You can specify a specific process ID using the `-p` parameter (or `--pid` parameter). The destination output directory must be specified using the `-d` parameter (or `--dump-dir` parameter).

Additionally, you can filter the DLLs you want to dump using a regular expression with the `--regex=REGEX` option and use `--ignore-case` if the match is case-insensitive.

1.6.3 `moddump` Plugin

This plugin works similarly to `dlldump`, but is specifically used to dump kernel drivers (called *modules*) to disk. You can filter the drivers you want to extract using a regular expression (`--regex=REGEX`) or specify the exact base address of the module with the `--base=BASE` parameter.

1.7 Command Console

1.7.1 `cmdscan` Plugin

The `cmdscan` plugin searches the memory of the `csrss.exe` process in Windows XP/2003/Vista/2008 or the `conhost.exe` process in Windows 7 for command line input that an attacker may have entered via a console (e.g., `cmd.exe`). This is especially useful when an attacker accessed the system through a remote desktop session or reverse shell and interacted with the command prompt.

The plugin locates internal console structures in memory that are not officially documented by Microsoft. These structures were identified through reverse engineering by researcher Michael Light, who analyzed the `conhost.exe` executable and the `winsrv.dll` library.

The output of `cmdscan` includes information about the console process each command is associated with, the name of the application that used the console, the console history memory address, the number of times the command was used, timestamps, and the process ID. It displays both active and previously executed commands.

```
➜  ~ vol2 -f /shared/zeus.vmem --profile=WinXPSP2x86 cmdscan
Volatility Foundation Volatility Framework 2.6.1
**************************************************
CommandProcess: csrss.exe Pid: 608
CommandHistory: 0xf786f8 Application: TPAutoConnect.exe Flags: Allocated
CommandCount: 0 LastAdded: -1 LastDisplayed: -1
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x448
```

1.7.2 `consoles` Plugin

The `consoles` plugin is similar to `cmdscan`, but uses a different internal structure to extract console-related data. Unlike `cmdscan`, it can also display the output of executed commands, not just the executed command string.

Additional information provided includes the console window title, the associated process name and ID, command aliases, and the screen coordinates of the `cmd.exe` console window.

```
➜  ~ vol2 -f /shared/zeus.vmem --profile=WinXPSP2x86 consoles
Volatility Foundation Volatility Framework 2.6.1
**************************************************
ConsoleProcess: csrss.exe Pid: 608
Console: 0x4e23b0 CommandHistorySize: 50
HistoryBufferCount: 1 HistoryBufferMax: 4
OriginalTitle: C:\Program Files\VMware\VMware Tools\TPAutoConnSvc.exe
Title: C:\Program Files\VMware\VMware Tools\TPAutoConnSvc.exe
AttachedProcess: TPAutoConnect.e Pid: 1084 Handle: 0x448
----
CommandHistory: 0xf786f8 Application: TPAutoConnect.exe Flags: Allocated
CommandCount: 0 LastAdded: -1 LastDisplayed: -1
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x448
----
Screen 0x4e2ab0 X:80 Y:25
Dump:
TPAutoConnect User Agent, Copyright (c) 1999-2009 ThinPrint AG, 7.17.512.1
**************************************************
ConsoleProcess: csrss.exe Pid: 608
Console: 0xf78958 CommandHistorySize: 50
HistoryBufferCount: 2 HistoryBufferMax: 4
OriginalTitle: ??ystemRoot%\system32\cmd.exe
Title:
```


1.8 Other Indicators of Compromise

1.8.1 `apihooks` Plugin

The `apihooks` plugin detects if a process has any API function hooks. A *hook* is a redirection of the execution flow, often used by malware to intercept or manipulate normal behavior. In this example (see below), process 856 has hooks to two functions in `ntdll.dll`, a core Windows library. The plugin also displays the first bytes of these hooks, which can help identify injected code.

```
➜  ~ vol2 -f /shared/zeus.vmem --profile=WinXPSP2x86 apihooks -p 856
Volatility Foundation Volatility Framework 2.6.1
**************************************************************************
Hook mode: Usermode
Hook type: Inline/Trampoline
Process: 856 (svchost.exe)
Victim module: ntdll.dll (0x7c900000 - 0x7c9b0000)
Function: ntdll.dll!NtCreateThread at 0x7c90d7d2
Hook address: 0xb73b47
Hooking module: <unknown>

Disassembly(0):
0x7c90d7d2 e970632684       JMP 0xb73b47
0x7c90d7d7 ba0003fe7f       MOV EDX, 0x7ffe0300
0x7c90d7dc ff12             CALL DWORD [EDX]
0x7c90d7de c22000           RET 0x20
0x7c90d7e1 90               NOP
0x7c90d7e2 90               NOP
0x7c90d7e3 90               NOP
0x7c90d7e4 90               NOP
0x7c90d7e5 90               NOP
0x7c90d7e6 90               NOP
0x7c90d7e7 b8               DB 0xb8
0x7c90d7e8 36               DB 0x36
0x7c90d7e9 00               DB 0x0

Disassembly(1):
0xb73b47 55               PUSH EBP
0xb73b48 8bec             MOV EBP, ESP
0xb73b4a 83ec18           SUB ESP, 0x18
```

1.8.2 `malfind` Plugin

The `malfind` plugin is one of the most powerful tools for detecting signs of memory injection. It looks for suspicious memory regions in user space marked as executable and writable, characteristics commonly used by injected shellcode or malware. It does not detect DLL injection using `LoadLibrary` or `CreateRemoteThread`, but instead looks for custom shellcode or executable regions that may have been manually allocated.

You can specify the `-D` option to save suspicious regions to disk. For each detection, metadata, a hexadecimal dump, and a disassembly fragment are displayed. After saving to disk, you can use the Linux command `file` to identify the type of dumped files.

```
➜  ~ vol2 -f /shared/zeus.vmem --profile=WinXPSP2x86 malfind -p 856 -D .
Volatility Foundation Volatility Framework 2.6.1
Process: svchost.exe Pid: 856 Address: 0xb70000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 38, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x0000000000b70000  4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00   MZ..............
0x0000000000b70010  b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00   ........@.......
0x0000000000b70020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x0000000000b70030  00 00 00 00 00 00 00 00 00 00 00 00 d0 00 00 00   ................

0x0000000000b70000 4d              DEC EBP
0x0000000000b70001 5a              POP EDX
0x0000000000b70002 90              NOP
0x0000000000b70003 0003            ADD [EBX], AL
0x0000000000b70005 0000            ADD [EAX], AL
0x0000000000b70007 000400          ADD [EAX+EAX], AL
0x0000000000b7000a 0000            ADD [EAX], AL
0x0000000000b7000c ff              DB 0xff
0x0000000000b7000d ff00            INC DWORD [EAX]
0x0000000000b7000f 00b800000000    ADD [EAX+0x0], BH
0x0000000000b70015 0000            ADD [EAX], AL
0x0000000000b70017 004000          ADD [EAX+0x0], AL
0x0000000000b7001a 0000            ADD [EAX], AL
0x0000000000b7001c 0000            ADD [EAX], AL
0x0000000000b7001e 0000            ADD [EAX], AL
0x0000000000b70020 0000            ADD [EAX], AL
0x0000000000b70022 0000            ADD [EAX], AL
0x0000000000b70024 0000            ADD [EAX], AL
```

```
➜  ~ ll
total 156K
-rw-r--r-- 1 forensic forensic 152K Mar 22 16:19 process.0x80ff88d8.0xb70000.dmp
-rw-r--r-- 1 forensic forensic 4.0K Mar 22 16:19 process.0x80ff88d8.0xcb0000.dmp
➜  ~ file *
process.0x80ff88d8.0xb70000.dmp: PE32 executable (GUI) Intel 80386, for MS Windows, 4 sections
process.0x80ff88d8.0xcb0000.dmp: DOS executable (COM), start instruction 0xb8350000 00e9cdd7
```

1.8.3 `yarascan` Plugin

The `yarascan` plugin allows you to scan memory using custom YARA rules. You can use the `--yara-file` option to load a rules file, or the `--yara-rules` parameter to define inline rules, byte patterns, or regular expressions (e.g., `--yara-rules="{eb 90 ff e4 88 32 0d }"` or `--yara-rules="/my(regular|expression{0,1})/"`).

By default, it scans user memory, but adding the `-K` option enables kernel space scanning. Note that the plugin expects a single `.yar` file. If you have multiple YARA rules files organized by family, you can use a Python script by Andrea Fortuna to merge them into a single file. A large collection of ready-to-use rules is available in the official YARA-Rules repository: `https://github.com/Yara-Rules`.

1.8.4 `svcscan` Plugin

The `svcscan` plugin lists the services registered in the system at the time of the memory acquisition. For each service, it provides: (i) associated process, (ii) whether the service is active, (iii) service type and current status, (iv) original and display names, (v) related executable file.

For instance, to identify malware that uses `svchost.exe` and hides its payload inside a DLL, use the `--verbose` option, which reveals the path to the `ServiceDLL` registry key:

```
Offset: 0x6eabb0
Order: 253
Start: SERVICE_AUTO_START
Process ID: 1028
Service Name: WZCSVC
Display Name: Wireless Zero Configuration
Service Type: SERVICE_WIN32_SHARE_PROCESS
Service State: SERVICE_RUNNING
Binary Path: C:\WINDOWS\System32\svchost.exe -k netsvcs
ServiceDll: %SystemRoot%\System32\wzcsvc.dll
ImagePath: %SystemRoot%\System32\svchost.exe -k netsvcs
FailureCommand:

Offset: 0x6eac40
Order: 254
Start: SERVICE_DEMAND_START
Process ID: -
Service Name: xmlprov
Display Name: Network Provisioning Service
Service Type: SERVICE_WIN32_SHARE_PROCESS
Service State: SERVICE_STOPPED
Binary Path: -
ServiceDll: %SystemRoot%\System32\xmlprov.dll
ImagePath: %SystemRoot%\System32\svchost.exe -k netsvcs
FailureCommand:
```

1.8.5 `ldrmodules` Plugin

The `ldrmodules` plugin detects hidden DLLs by comparing several internal Windows module lists that record the libraries loaded in a process. When a DLL is loaded into memory, Windows maintains pointers to it in multiple linked lists within the process's PEB (*Process Environment Block)* and loader data structures. These include:

- Load Order List: Lists modules in the order they were loaded.

- Memory Order List: Lists modules according to their location in memory.
- Initialization Order List: Lists modules in the order their entry points were called.

Under normal conditions, a loaded DLL will appear in all three lists. However, malware, especially advanced threats such as rootkits or fileless malware, may attempt to unlink a DLL from these lists to hide its presence. This is known as *DLL unlinking* or *DLL dumping*.

Even when a malicious DLL is removed from these lists, it typically still exists in memory. This is where Virtual Address Descriptors (VADs) come into play. VADs are kernel-level structures that Windows uses to manage the virtual memory allocation of each process. A VAD represents a range of memory assigned to a process and includes information about the base and end addresses of the memory region, access permissions (read, write, execute), whether the memory is assigned to a file (e.g., a DLL or an EXE), and the file's path on disk (if applicable).

Even if a DLL has been hidden in loader lists, its presence can be detected by traversing the process's VAD tree and checking for memory regions that are marked as executable, are assigned to a known DLL, and are not referenced in the usual module lists.

The `ldrmodules` plugin cross-references these different data sources to detect discrepancies, flagging modules that appear in one structure but not the others. These inconsistencies often point to attempts to hide code in memory and are strong indicators of compromise.

```
➜  ~ vol2 -f /shared/zeus.vmem --profile=WinXPSP2x86 ldrmodules
Volatility Foundation Volatility Framework 2.6.1
Pid      Process              Base       InLoad InInit InMem MappedPath
-------- -------------------- ---------- ------ ------ ----- ----------
     608 csrss.exe            0x75b60000 True   True   True  \WINDOWS\system32\winsrv.dll
     608 csrss.exe            0x77d40000 True   True   True  \WINDOWS\system32\user32.dll
     632 winlogon.exe         0x01000000 True   False  True  \WINDOWS\system32\winlogon.exe
     632 winlogon.exe         0x71ab0000 True   True   True  \WINDOWS\system32\ws2_32.dll
     632 winlogon.exe         0x7c900000 True   True   True  \WINDOWS\system32\ntdll.dll
     632 winlogon.exe         0x77d40000 True   True   True  \WINDOWS\system32\user32.dll
     632 winlogon.exe         0x7c9c0000 True   True   True  \WINDOWS\system32\shell32.dll
     632 winlogon.exe         0x76bf0000 True   True   True  \WINDOWS\system32\psapi.dll
     632 winlogon.exe         0x77b20000 True   True   True  \WINDOWS\system32\msasn1.dll
```

1.8.6 `idt` Plugin

The `idt` plugin prints the *Interrupt Descriptor Table* (IDT) for each CPU. The IDT is a fundamental structure of the operating system kernel that defines how the system responds to hardware and software interrupts, such as when a key is pressed, a page fault occurs, or a system call is made. Each IDT entry corresponds to a specific interrupt or exception and contains the address of the function to be called when that interrupt occurs (known as the *interrupt handler*).

In a healthy operating system, most IDT entries point to legitimate kernel routines located in trusted modules (e.g., the Windows kernel or HAL). However, rootkits (types of hidden malware) can manipulate the IDT to redirect interrupts or system calls to malicious code, effectively hiding their presence and gaining control over system behavior.

This plugin displays the address of the interrupt handler, the owning module, and the GDT selector. The *Global Descriptor Table* (GDT) is another critical structure used by the CPU to manage memory segmentation. It defines the base addresses, boundaries, and access rights for different memory segments (such as code, data, or system segments). Each GDT entry is identified by a selector, which the CPU uses to find the correct segment descriptor. The GDT plays an important role in defining the code execution environment, including its privilege level (kernel or user mode).

Think of the IDT as a "*routing table*" for interrupts: when an event occurs, the CPU consults the IDT to determine what to do. The GDT, on the other hand, is like a `blueprint` for memory segmentation, telling the CPU how to access different memory regions safely and efficiently.

Using `-v` (verbose) in this plugin adds additional details. This is useful for detecting rootkits that hijack IDT entries like `KiSystemService` to redirect system calls.

```
➜  ~ vol2 -f /shared/zeus.vmem --profile=WinXPSP2x86 idt
Volatility Foundation Volatility Framework 2.6.1
   CPU  Index   Selector Value      Module               Section
------ ------ ---------- ---------- -------------------- ------------
     0      0        0x8 0x8053d36c ntoskrnl.exe         .text
     0      1        0x8 0x8053d4e4 ntoskrnl.exe         .text
     0      2       0x58 0x00000000 NOT USED
     0      3        0x8 0x8053d8b4 ntoskrnl.exe         .text
     0      4        0x8 0x8053da34 ntoskrnl.exe         .text
     0      5        0x8 0x8053db90 ntoskrnl.exe         .text
     0      6        0x8 0x8053dd04 ntoskrnl.exe         .text
     0      7        0x8 0x8053e36c ntoskrnl.exe         .text
     0      8       0x50 0x00000000 NOT USED
     0      9        0x8 0x8053e790 ntoskrnl.exe         .text
```

1.8.7 `gdt` Plugin

Similar to the previous plugin, `gdt` displays the GDT, which rootkits can manipulate to allow user-mode programs to execute kernel-level code using constructs like `CALL FAR`.

```
➜  ~ vol2 -f /shared/zeus.vmem --profile=WinXPSP2x86 gdt
Volatility Foundation Volatility Framework 2.6.1
   CPU        Sel Base       Limit      Type             DPL Gr   Pr
------ ---------- ---------- ---------- -------------- ------ ---- ----
     0        0x0 0x00000000 0x00000000 <Reserved>          0 By   Np
     0        0x8 0x00000000 0xffffffff Code RE Ac          0 Pg   P
     0       0x10 0x00000000 0xffffffff Data RW Ac          0 Pg   P
     0       0x18 0x00000000 0xffffffff Code RE Ac          3 Pg   P
     0       0x20 0x00000000 0xffffffff Data RW Ac          3 Pg   P
     0       0x28 0x80042000 0x000020ab TSS32 Busy          0 By   P
     0       0x30 0xffdff000 0x00001fff Data RW Ac          0 Pg   P
```

1.8.8 `callbacks` plugin

This plugin detects kernel callbacks and notification routines registered by the system, anti-malware software, or rootkits. These include:

- `PsSetCreateProcessNotifyRoutine` (process creation),
- `PsSetCreateThreadNotifyRoutine` (thread creation),

- `PsSetImageLoadNotifyRoutine` (image load),
- `IoRegisterFsRegistrationChange` (file system registry),
- `KeRegisterBugCheck`, `KeRegisterBugCheckReasonCallback`,
- `CmRegisterCallback` (registration callbacks on Windows XP),
- `CmRegisterCallbackEx` (registration callbacks on Windows Vista and 7),
- `IoRegisterShutdownNotification` (shutdown callbacks),
- `DbgSetDebugPrintCallback` (debug print callbacks on Windows Vista and 7), and
- `DbgkLkmdRegisterCallback` (debug callbacks in Windows 7).

```
➜  ~ vol2 -f /shared/zeus.vmem --profile=WinXPSP2x86 callbacks
Volatility Foundation Volatility Framework 2.6.1
Type                                Callback   Module              Details
----------------------------------- ---------- ------------------- -------
IoRegisterShutdownNotification      0xfc9af5be Fs_Rec.SYS          \FileSystem\Fs_Rec
IoRegisterShutdownNotification      0xfc9af5be Fs_Rec.SYS          \FileSystem\Fs_Rec
IoRegisterShutdownNotification      0xf3b457fa vmhgfs.sys          \FileSystem\vmhgfs
IoRegisterShutdownNotification      0xfc0f765c VIDEOPRT.SYS        \Driver\mnmdd
IoRegisterShutdownNotification      0xfc0f765c VIDEOPRT.SYS        \Driver\VgaSave
IoRegisterShutdownNotification      0xfc6bec74 Cdfs.SYS            \FileSystem\Cdfs
IoRegisterShutdownNotification      0xfc9af5be Fs_Rec.SYS          \FileSystem\Fs_Rec
IoRegisterShutdownNotification      0xfc9af5be Fs_Rec.SYS          \FileSystem\Fs_Rec
IoRegisterShutdownNotification      0xfc9af5be Fs_Rec.SYS          \FileSystem\Fs_Rec
IoRegisterShutdownNotification      0xfc0f765c VIDEOPRT.SYS        \Driver\vmx_svga
IoRegisterShutdownNotification      0xfc0f765c VIDEOPRT.SYS        \Driver\RDPCDD
IoRegisterShutdownNotification      0xfc33d2be ftdisk.sys          \Driver\Ftdisk
IoRegisterShutdownNotification      0xfc1db33d Mup.sys             \FileSystem\Mup
IoRegisterShutdownNotification      0x805f4630 ntoskrnl.exe        \Driver\WMIxWDM
IoRegisterShutdownNotification      0x805cc77c ntoskrnl.exe        \FileSystem\RAW
IoRegisterFsRegistrationChange      0xfc2c0876 sr.sys              -
IoRegisterShutdownNotification      0xfc4ab73a MountMgr.sys        \Driver\MountMgr
GenericKernelCallback               0xfc58e194 vmci.sys            -
PsSetCreateProcessNotifyRoutine     0xfc58e194 vmci.sys            -
KeBugCheckCallbackListHead          0xfc1e85ed NDIS.sys            Ndis miniport
KeBugCheckCallbackListHead          0x806d57ca hal.dll             ACPI 1.0 - APIC platform UP
KeRegisterBugCheckReasonCallback    0xfc967ac0 mssmbios.sys        SMBiosDa
KeRegisterBugCheckReasonCallback    0xfc967a78 mssmbios.sys        SMBiosRe
KeRegisterBugCheckReasonCallback    0xfc967a30 mssmbios.sys        SMBiosDa
KeRegisterBugCheckReasonCallback    0xfc0d5006 USBPORT.SYS         USBPORT
KeRegisterBugCheckReasonCallback    0xfc0d4f66 USBPORT.SYS         USBPORT
KeRegisterBugCheckReasonCallback    0xfc0eb3e2 VIDEOPRT.SYS        -
```

1.8.9 `driverirp` Plugin

The `driverirp` plugin lists the I/O Request Packet (IRP) functions for each driver. Prints the address, function purpose, and owning module. You can also detect hooks in these functions and disassemble their code with `-v`.

1.8.10 `devicetree` Plugin

This plugin displays the relationships between device objects and drivers. It is useful for identifying hidden or unauthorized devices that might be linked to a rootkit.

1.8.11 `timers` Plugin

This plugin lists kernel timers and associated Deferred Procedure Calls (DPCs). Some rootkits use these timers to periodically execute malicious actions. Suspicious timers pointing to unknown kernel memory regions may indicate malicious behavior and should be further investigated using disassembly or known IoCs.

```
➜  ~ vol2 -f /shared/zeus.vmem --profile=WinXPSP2x86 timers
Volatility Foundation Volatility Framework 2.6.1
Offset(V)  DueTime                  Period(ms) Signaled   Routine    Module
---------- ---------------------- ---------- ---------- ---------- ------
0xff265568 0x00000001:0x01a8e254            0 -          0x80534016 ntoskrnl.exe
0xff12d370 0x80000000:0xe42c8d48            0 -          0x80534016 ntoskrnl.exe
0x8055a400 0x00003c13:0x3f3c8118            0 -          0x80533b58 ntoskrnl.exe
0x8055a380 0x006434d7:0x637f9828            0 -          0x80533b7e ntoskrnl.exe
0x8055a300 0x00000008:0x61fb3e16            0 -          0x80533bf8 ntoskrnl.exe
0xf3b1f320 0x00000000:0xf5dd5c48            0 -          0xf3b15385 rdbss.sys
0xf3bf1910 0x00000000:0xf5e1d7be          100 Yes        0xf3ba93dd tcpip.sys
0xff3d4730 0x00000000:0xf5e5a84d            0 -          0xfc0cc4ec USBPORT.SYS
0x80ee1730 0x00000000:0xf5e80aa7            0 -          0xfc0cc4ec USBPORT.SYS
0x80550a00 0x00000000:0xf5e80aa8         1000 Yes        0x804f33da ntoskrnl.exe
0x805508d0 0x00000000:0xfaacbea8        60000 Yes        0x804f3b72 ntoskrnl.exe
0xff39e6b0 0x00000001:0x0452c2e0        30000 Yes        0xf3b5f385 afd.sys
0xff27bcd8 0x00000001:0x7200e0ec            0 -          0x80534016 ntoskrnl.exe
0xffafb188 0x00000000:0xf60bcdee            0 -          0xfc2b592e sr.sys
0x8054f288 0x00000000:0xf612f4fc            0 -          0x804e5aec ntoskrnl.exe
0xff1bbdf8 0x00000000:0xf61c7e64            0 -          0xfc7933f0 TDI.SYS
0xff375b38 0x00000000:0xf61c7e64            0 -          0xf3b6d240 afd.sys
```

1.8.12 `getsids` Plugin

The `getsids` plugin displays the security identifiers (SIDs) for each process. This helps identify which user account the process belongs to and whether it has escalated privileges (e.g., at the system level).


```
➜  ~ vol2 -f /shared/zeus.vmem --profile=WinXPSP2x86 getsids
Volatility Foundation Volatility Framework 2.6.1
System (4): S-1-5-18 (Local System)
System (4): S-1-5-32-544 (Administrators)
System (4): S-1-1-0 (Everyone)
System (4): S-1-5-11 (Authenticated Users)
smss.exe (544): S-1-5-18 (Local System)
smss.exe (544): S-1-5-32-544 (Administrators)
smss.exe (544): S-1-1-0 (Everyone)
smss.exe (544): S-1-5-11 (Authenticated Users)
csrss.exe (608): S-1-5-18 (Local System)
csrss.exe (608): S-1-5-32-544 (Administrators)
csrss.exe (608): S-1-1-0 (Everyone)
csrss.exe (608): S-1-5-11 (Authenticated Users)
winlogon.exe (632): S-1-5-18 (Local System)
winlogon.exe (632): S-1-5-32-544 (Administrators)
winlogon.exe (632): S-1-1-0 (Everyone)
winlogon.exe (632): S-1-5-11 (Authenticated Users)
```

1.8.13 `privs` Plugin

This plugin displays the privileges assigned to each process. The `--silent` option displays only enabled privileges, while `--regex` allows filtering by privilege name. This is useful for detecting processes that may bypass security controls.

```
➜  ~ vol2 -f /shared/zeus.vmem --profile=WinXPSP2x86 privs -p 856
Volatility Foundation Volatility Framework 2.6.1
Pid      Process          Value  Privilege                         Attributes              Description
-------- ---------------- ------ --------------------------------- ----------------------- -----------
     856 svchost.exe           7 SeTcbPrivilege                    Present,Enabled,Default  Act as part of the operating system
     856 svchost.exe           2 SeCreateTokenPrivilege            Present                  Create a token object
     856 svchost.exe           9 SeTakeOwnershipPrivilege          Present                  Take ownership of files/objects
     856 svchost.exe          15 SeCreatePagefilePrivilege         Present,Enabled,Default  Create a pagefile
     856 svchost.exe           4 SeLockMemoryPrivilege             Present,Enabled,Default  Lock pages in memory
     856 svchost.exe           3 SeAssignPrimaryTokenPrivilege     Present                  Replace a process-level token
     856 svchost.exe           5 SeIncreaseQuotaPrivilege          Present                  Increase quotas
     856 svchost.exe          14 SeIncreaseBasePriorityPrivilege   Present,Enabled,Default  Increase scheduling priority
     856 svchost.exe          16 SeCreatePermanentPrivilege        Present,Enabled,Default  Create permanent shared objects
     856 svchost.exe          20 SeDebugPrivilege                  Present,Enabled,Default  Debug programs
     856 svchost.exe          21 SeAuditPrivilege                  Present,Enabled,Default  Generate security audits
     856 svchost.exe           8 SeSecurityPrivilege               Present                  Manage auditing and security log
     856 svchost.exe          22 SeSystemEnvironmentPrivilege      Present                  Edit firmware environment values
     856 svchost.exe          23 SeChangeNotifyPrivilege           Present,Enabled,Default  Receive notifications of changes to files or directories
     856 svchost.exe          17 SeBackupPrivilege                 Present                  Backup files and directories
     856 svchost.exe          18 SeRestorePrivilege                Present                  Restore files and directories
     856 svchost.exe          19 SeShutdownPrivilege               Present                  Shut down the system
     856 svchost.exe          10 SeLoadDriverPrivilege             Present                  Load and unload device drivers
     856 svchost.exe          13 SeProfileSingleProcessPrivilege   Present,Enabled,Default  Profile a single process
     856 svchost.exe          12 SeSystemtimePrivilege             Present                  Change the system time
     856 svchost.exe          25 SeUndockPrivilege                 Present                  Remove computer from docking station
     856 svchost.exe          28 SeManageVolumePrivilege           Present                  Manage the files on a volume
     856 svchost.exe          29 SeImpersonatePrivilege            Present,Enabled,Default  Impersonate a client after authentication
     856 svchost.exe          30 SeCreateGlobalPrivilege           Present,Enabled,Default  Create global objects
```


1.8.14 `verinfo`

The `verinfo` plugin extracts metadata from executables and DLLs in user space. This includes version information, product name, company, and description. Please note that malware authors often forge or remove this data, so it is not always reliable.

```
➜  ~ vol2 -f /shared/zeus.vmem --profile=WinXPSP2x86 verinfo
Volatility Foundation Volatility Framework 2.6.1
C:\WINDOWS\system32\winsrv.dll
  File version    : 5.1.2600.2180
  Product version : 5.1.2600.2180
  Flags           :
  OS              : Windows NT
  File Type       : Dynamic Link Library
  File Date       :
  CompanyName : Microsoft Corporation
  FileDescription : Windows Server DLL
  FileVersion : 5.1.2600.2180 (xpsp_sp2_rtm.040803-2158)
  InternalName : winsrv
  LegalCopyright : \xa9 Microsoft Corporation. All rights reserved.
  OriginalFilename : winsrv.dll
  ProductName : Microsoft\xae Windows\xae Operating System
  ProductVersion : 5.1.2600.2180
C:\WINDOWS\system32\USER32.dll
  File version    : 5.1.2600.2180
  Product version : 5.1.2600.2180
  Flags           :
  OS              : Windows NT
  File Type       : Dynamic Link Library
  File Date       :
  CompanyName : Microsoft Corporation
  FileDescription : Windows XP USER API Client DLL
  FileVersion : 5.1.2600.2180 (xpsp_sp2_rtm.040803-2158)
  InternalName : user32
  LegalCopyright : \xa9 Microsoft Corporation. All rights reserved.
  OriginalFilename : user32
  ProductName : Microsoft\xae Windows\xae Operating System
  ProductVersion : 5.1.2600.2180
\??\C:\WINDOWS\system32\winlogon.exe
C:\WINDOWS\system32\ntdll.dll
C:\WINDOWS\system32\kernel32.dll
C:\WINDOWS\system32\ADVAPI32.dll
C:\WINDOWS\system32\RPCRT4.dll
C:\WINDOWS\system32\msvcrt.dll
```


1.8.15 `envars` Plugin

This plugin lists the environment variables for each process. It includes useful context such as the system path, current directory, temporary folders, username, hostname, number of processors, etc.

```
➜  ~ vol2 -f /shared/zeus.vmem --profile=WinXPSP2x86 envars -p 856
Volatility Foundation Volatility Framework 2.6.1
Pid      Process            Block      Variable                       Value
-------- ------------------ ---------- ------------------------------ -----
     856 svchost.exe        0x00010000 ALLUSERSPROFILE                C:\Documents and Settings\All Users
     856 svchost.exe        0x00010000 CommonProgramFiles             C:\Program Files\Common Files
     856 svchost.exe        0x00010000 COMPUTERNAME                   BILLY-DB5B96DD3
     856 svchost.exe        0x00010000 ComSpec                        C:\WINDOWS\system32\cmd.exe
     856 svchost.exe        0x00010000 FP_NO_HOST_CHECK               NO
     856 svchost.exe        0x00010000 NUMBER_OF_PROCESSORS           1
     856 svchost.exe        0x00010000 OS                             Windows_NT
     856 svchost.exe        0x00010000 Path                           C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem
     856 svchost.exe        0x00010000 PATHEXT                        .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH
     856 svchost.exe        0x00010000 PROCESSOR_ARCHITECTURE         x86
     856 svchost.exe        0x00010000 PROCESSOR_IDENTIFIER           x86 Family 6 Model 23 Stepping 10, GenuineIntel
     856 svchost.exe        0x00010000 PROCESSOR_LEVEL                6
     856 svchost.exe        0x00010000 PROCESSOR_REVISION             170a
     856 svchost.exe        0x00010000 ProgramFiles                   C:\Program Files
     856 svchost.exe        0x00010000 SystemDrive                    C:
     856 svchost.exe        0x00010000 SystemRoot                     C:\WINDOWS
     856 svchost.exe        0x00010000 TEMP                           C:\WINDOWS\TEMP
     856 svchost.exe        0x00010000 TMP                            C:\WINDOWS\TEMP
     856 svchost.exe        0x00010000 USERPROFILE                    C:\WINDOWS\system32\config\systemprofile
     856 svchost.exe        0x00010000 windir                         C:\WINDOWS
```

2 What's next?

After completing this lab, we recommend practicing with other memory dumps (e.g., those from modern systems like Windows 10), exploring Volatility 3 in the following lab, or combining memory forensics with disk analysis using tools like Autopsy or Velociraptor.

