First steps with Volatility

In this lab you will be introduced to malware forensics with Volatility. To do this, you are going to make use of the virtual machine provided by the instructor along with one of the memory dumps. All this additional material is available on the web page of this training workshop: `https://webdiis.unizar.es/~ricardo/sbc-2022/malware-memory-forensics/`. First, the laboratory environment that will be used in this course is explained, and finally, an analysis of a memory dump is carried out as an example to understand the operation of the Volatility memory dump analysis environment, presenting a series of plugins that come with Volatility and the results of execution.

1 Laboratory environment

1.1 Deployment of the virtual machine

The virtual machine required for this training workshop, *Cybersecurity Summer BootCamp 2022* is provided in OVA format, so you can deploy it in VirtualBox or any other hypervisor software of your choice.

To deploy it in VirtualBox, go to the menu File > Import Appliance... (see Figure 1). In the new window, select the OVA file that will have been created after decompressing the compressed file available on the workshop website, by pressing the Continue button in the window. In the window that appears below you can define other aspects of the virtual machine, such as its default configuration or the path where the virtual machine will be stored. By default, a minimum of 4GiB of virtual memory has been set for smooth execution of the virtual machine. If you need to adjust it, you can do it right here (if you have a lot of RAM you can even increase this amount a bit). Once you have finished the configuration, press the Import button (see Figure 2).

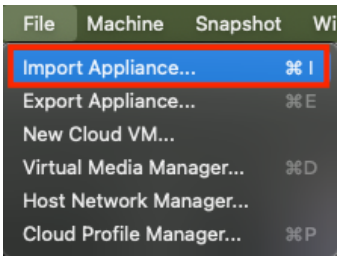

Figure 1: Menu File > Import Appliance... in VirtualBox.

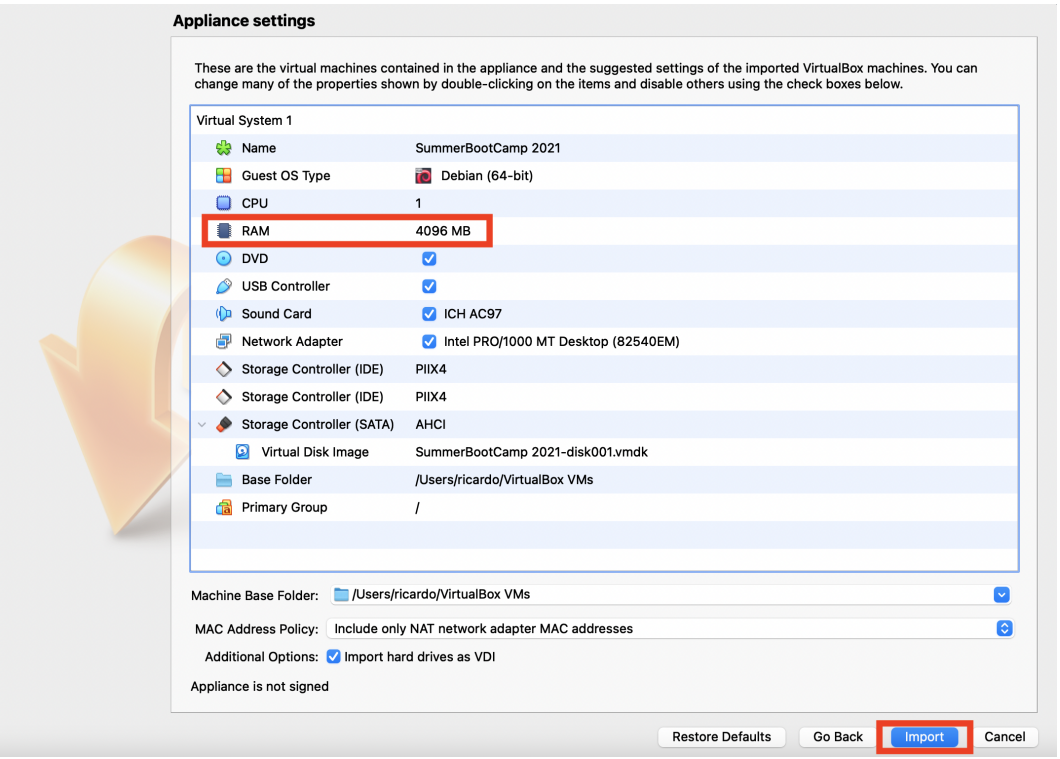

Figure 2: Configuration of the imported virtual machine.

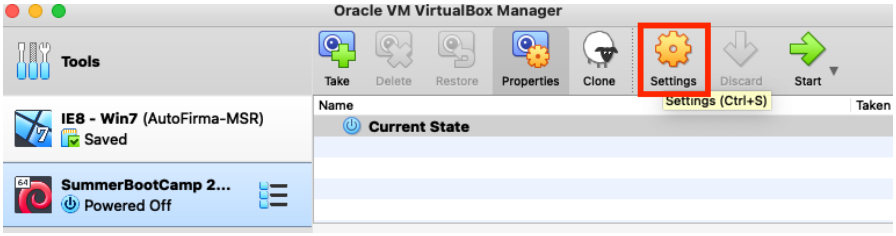

Figure 3: Settings button for changing imported virtual machine settings.

If you don't want to use the virtual machine and want to use your own machine for the training workshop, that is also possible. You simply need to have the following basic software installed:

- Python 2.7 and Python 3.7
- Volatility 2.6 and Volatility 3, available in their GitHub repositories:
  - `https://github.com/volatilityfoundation/volatility`
  - `https://github.com/volatilityfoundation/volatility3`
- Integrated Development Environment of your choice (Visual Studio Code is installed on the virtual machine).

The rest of the material (basically, software from GitHub repositories) will be installed as needed during the lab sessions. For the virtual machine user, you can use the user account `alumno` (password `alumno`). The root account (password `toor`) **should only be used when strictly necessary**.

1.2 Shared folder between host machine and virtual machine

The size of a memory dump depends, logically, on the RAM memory of the machine where the capture was made. In order not to overload the virtual machine's hard drive, it is recommended that a folder be shared between the host machine and the virtual machine. To do this, follow these steps:

1. Select the imported virtual machine settings using the Settings button (see Figure 3).
2. Select the folder you want to share with the virtual machine and press the Ok button (see Figure 4).

Now, every time you run the virtual machine you will have to execute a command from the Linux terminal inside the virtual machine:

```
$ su (contraseña de root)
# mount -t vboxsf sharedVM /mnt
```

Notice that the type of files (`-t` parameter of the `mount` command) is `vboxsf`. Next, you have to indicate the shared folder name given in the shared folder selection window (Folder Name field, see Figure 4). After this, if you access the folder `/mnt` you will see that you have access to the shared folder from the virtual machine.

Continue the rest of the lab session with the user account of `alumno`.

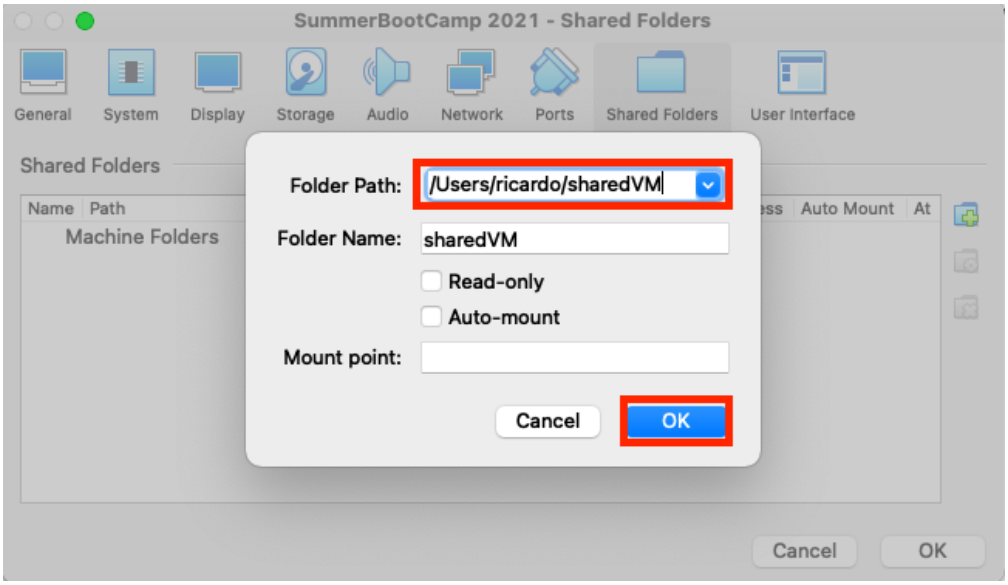

Figure 4: Shared folder selection between virtual machine and host machine.

2 `zeus.vmem` Dump Analysis

As an example of the use of Volatility (specifically, Volatility version 2.6.1), we are going to analyze a memory dump of a Windows machine infected with the famous ZeuS banking Trojan. This dump is available on the workshop website (`https://webdiis.unizar.es/~ricardo/sbc-2022/malware-memory-forensics/additional/dumps/zeus.vmem.zip`) and comes from *Malware Analyst's Cookbook*.

2.1 Recognizing the origin of the dump

Volatility 2.6 needs to know the operating system the dump to be analyzed in order to know where the kernel structures of interest are located to accurately analyze the dump. The plugin for this is called `imageinfo`. So, in this example we can execute the following command:

```
$ python2 vol.py -f ~/volcados/zeus.vmem imageinfo
Volatility Foundation Volatility Framework 2.6.1
INFO    : volatility.debug    : Determining profile based on KDBG search...
          Suggested Profile(s) : WinXPSP2x86, WinXPSP3x86 (Instantiated with WinXPSP2x86)
                     AS Layer1 : IA32PagedMemoryPae (Kernel AS)
                     AS Layer2 : FileAddressSpace (/Users/ricardo/volcados/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 seen in the output provided, the recommended profile is either `WinXPSP2x86` or `WinXPSP3x86`.

2.2 Running processes

The Volatility plugin that allows us to know which processes were running in the analyzed dump is `pslist`. This plugin walks through the double linked list pointed to by `PsActiveProcessHead` (internal structure of the Windows kernel) and shows for each process its offset, process name, identifier, parent process identifier, number of threads, number of *handles*, session identifier, if it is a WoW64 process (will only appear in dumps from 64-bit machines), and the date and time the process started and/or finished execution.

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

```
[9:02:49] ricardo:volatility git:(master) $ python2 vol.py -f ~/volcados/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
```

Those processes that are hidden or have been unlinked from the process list will not be shown by the output of this plugin. There is another plugin, however, that does allow you to show these types of hidden or evasive processes: `psscan`.

```
[9:03:11] ricardo:volatility git:(master) $ python2 vol.py -f ~/volcados/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
```

Another plugin of interest related to processes is `pstree`. This plugin shows a tree of all the processes captured in the dump, showing the relationships between them.


```
[9:26:43] ricardo:volatility git:(master) $ python2 vol.py -f ~/volcados/zeus.vmem --profile=WinXPSP2x86 pstre
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
```

Finally, the plugin `psxview` also allows detecting possible hidden processes, as it compares the results of process searches using various techniques (`PsActiveProcessHead` list, `EPROCESS` objects, `ETHREAD` objects, list `PspCidTable` and by `csrss.exe`).

```
[12:50:53] ricardo:volatility git:(master) $ python2 vol.py -f ~/volcados/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
```

Related to processes, the `dlllist` plugin allows you to check which function libraries are linked to each of the processes found in the dump. You can filter for a particular process with the `-p`

parameter.

```
[13:09:24] ricardo:volatility git:(master) $ python2 vol.py -f ~/volcados/zeus.vmem --profile=WinXPSP2x86 dlllist
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
rDll=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
```

Similarly, the plugin `modscan` allows you to view the drivers that were running at the time of the memory dump acquisition.

```
[13:28:39] ricardo:volatility git:(master) $ python2 vol.py -f ~/volcados/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\Drivers\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
0x000000000010ad638 raspti.sys         0xfc7a3000     0x5000 \SystemRoot\system32\DRIVERS\raspti.sys
```

Finally, the `threads` plugin allows you to know details about the threads, including the content of the processor registers of each of them, a small disassembly of their start address and other fields that may be of interest for a more detailed investigation.


```
[12:49:06] ricardo:volatility git:(master) $ python2 vol.py -f ~/volcados/zeus.vmem --profile=WinXPSP2x86 threads
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
0x7c810862 90               NOP
```

By default, this command displays information for all threads in the system. If you want to filter, you can use the `-F` (or `--filter`) parameter. One or more filters can be specified, separated by commas. The possible possible filters can be consulted with the option `-L`:

```
[12:50:02] ricardo:volatility git:(master) $ python2 vol.py -f ~/volcados/zeus.vmem --profile=WinXPSP2x86 threads -L
Volatility Foundation Volatility Framework 2.6.1
Impersonation        Detect impersonating threads
ScannerOnly          Detect threads no longer in a linked list
DkomExit             Detect inconsistencies wrt exit times and termination
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
```

2.3 Internet connection

The `connscan` plugin looks for `_TCPT_OBJECT` objects, which represent Internet connection objects, both open (at memory acquisition time) and recently terminated. The memory address of the object and the local and remote addresses of the connection are displayed, as well as the process identifier associated with the connection. This last field may not be retrieved correctly, so false positives may appear. This command only works for Windows XP and Windows 2003 server versions.


```
[9:12:30] ricardo:volatility git:(master) $ python2 vol.py -f ~/volcados/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
```

It can be seen that two connections related to the process with identifier 856 appear. According to what has been seen before in Section 2.2, this identifier corresponds to a process `svchost.exe`, whose parent is the process `services.exe` (ID 676).

To display only the objects related to the active connections at the time of acquisition, you can use the `connections` plugin. Another plugin of interest is `sockets`, which displays information about all sockets of any type (TCP, UDP, RAW, etc.). In current versions of Windows this command may not work properly. This command only works for Windows XP and Windows 2003 Server versions.

```
[9:18:11] ricardo:volatility git:(master) $ python2 vol.py -f ~/volcados/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
```

In the case of newer versions of Windows (such as Windows 7) you have to use the plugin `netscan`. We will see this plugin in operation later.

2.4 Files

To consult the files related to the processes, the plugin `handles` can be used. This plugin accepts a `-t` parameter where we can specify the type of *handle* we are interested in. For example, if we are interested in seeing the mutual exclusion objects (mutex) associated with the process from which we had previously detected some connections (process with identifier 856), we can consult the objects of type *Mutant* (in Windows, the mutex objects are called *mutants*).


```
[9:53:12] ricardo:volatility git:(master) $ python2 vol.py -f ~/volcados/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 settings!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 settings!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
```

Notice that there is a mutex with the name `_AVIRA_`. This mutex name is the standard mutex created by the Zeus banking Trojan to indicate that it is present on the system. Early versions of Zeus used the `winlogon.exe` process as the process to remain on the infected machine.

We are now going to check the mutexes that the process `winlogon.exe` (identifier 632) has open, using `-t Mutant` and filtering using the `grep` command:

```
0xff1e68b0    856      0x460   0x100000 Mutant           RasPbFile
[9:53:21] ricardo:volatility git:(master) $ python2 vol.py -f ~/volcados/zeus.vmem --profile=WinXPSP2x86 handles -t Mutant -p 632 | grep
AVIRA
Volatility Foundation Volatility Framework 2.6.1
0xff1e7dc0    632      0x8bc   0x1f0001 Mutant           _AVIRA_2109
```

As you can see, there is also an object of type mutex with the name `_AVIRA_`. If the files associated with this process are now searched for `winlogon.exe` (parameter `-t File`):

```
[9:43:58] ricardo:volatility git:(master) $ python2 vol.py -f ~/volcados/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.Common-Controls_6595b64144ccf1df_6.0.2600.2180_x-ww_a84f1ff9
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.Common-Controls_6595b64144ccf1df_6.0.2600.2180_x-ww_a84f1ff9
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 Shared\web server extensions\40\isapi\_vti_adm
0xff21ff90    632      0x210   0x160001 File             \Device\HarddiskVolume1\Program Files\Common Files\Microsoft Shared\web server extensions\40\_vti_bin\_vti_adm
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 Shared\web server extensions\40\isapi\_vti_aut
0xff242aa8    632      0x220   0x160001 File             \Device\HarddiskVolume1\Program Files\Common Files\Microsoft Shared\web server extensions\40\_vti_bin\_vti_aut
0xff24a3f0    632      0x224   0x160001 File             \Device\HarddiskVolume1\WINDOWS\system32\inetsrv
0xff26f8d8    632      0x228   0x160001 File             \Device\HarddiskVolume1\Program Files\Common Files\Microsoft Shared\web server extensions\40\bin
0x80f60b68    632      0x604   0x12019f File             \Device\NamedPipe\SfcApi
0xff3cab58    632      0x608   0x12019f File             \Device\NamedPipe\SfcApi
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.Common-Controls_6595b64144ccf1df_6.0.2600.2180_x-ww_a84f1ff9
0xff135930    632      0x810   0x12019f File             \Device\KSENUM#00000001\{98365890-165F-11D0-A195-0020AFD156E4}
0xff3af028    632      0x83c   0x12019f File             \Device\NamedPipe\winlogonrpc
0x80f5cd78    632      0x898   0x12019f File             \Device\NamedPipe\_AVIRA_2109
```

Looking at the output of the command, you can see references to files such as `user.ds`, `local.ds` and the binary file `sdra64.exe`. The files `user.ds` and `local.ds` contain the configu-

ration and information stolen by the Zeus Trojan, while the binary file `sdra64.exe` is the Zeus installer. Finally, also notice that there is a pipe with the same name as the mutex. Named pipes are used to communicate between processes or to redirect command output to files on disk.

2.5 Windows Registry

The Windows Registry contains both volatile information (dynamically generated on each run) and static information. During Windows startup, a series of files are loaded from disk to build the Windows Registry, which is necessary for the operating system to function properly.

To know where the files related to the different parts of the Windows Registry are located, you can use the `hivelist` plugin:

```
[10:03:26] ricardo:volatility git:(master) $ python2 vol.py -f ~/volcados/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\UsrCl
ass.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\Usr
Class.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\UsrC
lass.dat
0xe1da4008 0x00f6e008 \Device\HarddiskVolume1\Documents and Settings\Administrator\NTUSER.DAT
```

Note that the last column shows the information related to the disk files as well as where each of the Registry keys are located. The file `ntuser.dat` is the one that makes up the key `HKCU` of the Registry.

If you would like to see all the subkeys of a specific key, you can use the `hivedump` command, passing it the virtual address of the key you want to query (`-o` parameter):

```
[10:23:04] ricardo:volatility git:(master) $ python2 vol.py -f ~/volcados/zeus.vmem --profile=WinXPSP2x86 hivedump -o 0xe1da4008
Volatility Foundation Volatility Framework 2.6.1
Last Written         Key
2010-08-11 06:06:48 UTC+0000 \$$$PROTO.HIV
2010-06-10 16:11:42 UTC+0000 \$$$PROTO.HIV\AppEvents
2010-06-10 16:12:07 UTC+0000 \$$$PROTO.HIV\AppEvents\EventLabels
2010-06-10 16:11:42 UTC+0000 \$$$PROTO.HIV\AppEvents\EventLabels\.Default
2010-06-10 16:12:00 UTC+0000 \$$$PROTO.HIV\AppEvents\EventLabels\ActivatingDocument
2010-06-10 16:11:42 UTC+0000 \$$$PROTO.HIV\AppEvents\EventLabels\AppGPFault
2010-06-10 16:12:00 UTC+0000 \$$$PROTO.HIV\AppEvents\EventLabels\BlockedPopup
2010-06-10 16:12:00 UTC+0000 \$$$PROTO.HIV\AppEvents\EventLabels\CCSelect
2010-06-10 16:11:42 UTC+0000 \$$$PROTO.HIV\AppEvents\EventLabels\Close
2010-06-10 16:11:42 UTC+0000 \$$$PROTO.HIV\AppEvents\EventLabels\CriticalBatteryAlarm
```

Finally, the plugin `printkey` allows us to query a particular key from the Windows Registry. Note that it could be the case that the key to be queried was not present in memory at the time of acquisition and, therefore, it would not be possible to obtain its value. In this case, consider that you want to see if the machine had Windows Firewall enabled by looking at the registry key `ControlSet001\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile`:


```
[9:48:28] ricardo:volatility git:(master) $ python2 vol.py -f ~/volcados/zeus.vmem --profile=WinXPSP2x86 printkey -K 'Control
Set001\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 can be seen, the output of the command shows that the Windows Firewall is not activated.

2.6 Memory artifact dump

2.6.1 `procdump` plugin

This plugin allows you to dump the process of an executable to disk. It admits the parameter `-u` (or `--unsafe`) to avoid certain checks when parsing the header of an executable. Some malware samples can modify PE headers so that dump tools fail.

Another parameter of interest is `--memory`, which is a more accurate view of the content in memory. With this parameter, the process is dumped as it is in memory, considering all the additional space for memory alignment added by the operating system when loading the executable.

2.6.2 `dlldump` plugin

The `dlldump` plugin allows you to dump the libraries contained in a memory dump to disk files. A particular process id can be specified using the `-p` (or `--pid` parameter), and requires the `-D` (or `--dump-dir` parameter) to specify the directory where the modules extracted from the dump will be dumped. You can also dump those libraries that meet a certain regular expression (`--regex=REGEX`), ignoring or not case (`--ignore-case`).

2.6.3 `moddump` plugin

This plugin is similar to the previous one, but it is used to dump the kernel drivers of the operating system to disk. You can specify which driver to extract from the dump by using a regular expression (`--regex=REGEX`) or by specifying the base address (`--base=BASE`).

2.7 Command console

2.7.1 `cmdscan` plugin

The `cmdscan` plugin searches the memory of the process `csrss.exe` on Windows XP/2003/Vista/2008 or the process `conhost.exe` on Windows 7 for commands that may have been entered by attackers through a console (file `cmd.exe`). This command is useful to find out what an attacker has been able to do, whether it is opening a console through an RDP session or a reverse console.

The search method carried out by the plugin is based on locating certain structures within the process under analysis. These structures, unlike other Windows operating system structures, are not public and were achieved by reverse engineering both the executable file `conhost.exe` and the library `winsrv.dll` (performed by the researcher Michael Light).

The output of this command shows which console process it belongs to, the name of the application using the console, where the displayed console history structure is located, how many times this command has been used, the last time it was added, and displayed and the process handler. This plugin shows both active and finished console commands.

```
[16:39:20] ricardo:volatility git:(master) $ python2 vol.py -f ~/volcados/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
```

2.7.2 `consoles` plugin

This plugin is similar to the previous one, although it looks for command information through another internal structure. Unlike the previous one, it will not only show information about the command that was executed, but also the result of said command.

In addition, it provides information about the title of the window, the name and the identifier of the associated process, if the executed command has some kind of alias, and the screen coordinates of the console `cmd.exe`.

```
[16:47:57] ricardo:volatility git:(master) $ python2 vol.py -f ~/volcados/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:
```

2.8 Other indicators of compromise

2.8.1 `apihooks` plugin

The `apihooks` plugin lets you know if a process has some type of hook. A *hook* is defined as an alteration of the normal flow of execution of the application, diverting its execution to another place. Querying the process with identifier 856, it is observed that it has two hooks in two functions of `ntdll`. Also, this command shows us the first bytes of these hooks:


```
[10:26:38] ricardo:volatility git:(master) $ python2 vol.py -f ~/volcados/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
0xb73b4d 53                 PUSH EBX
0xb73b4e 56                 PUSH ESI
0xb73b4f 57                 PUSH EDI
0xb73b50 8b7d14             MOV EDI, [EBP+0x14]
0xb73b53 8d4514             LEA EAX, [EBP+0x14]
0xb73b56 50                 PUSH EAX
0xb73b57 6a18               PUSH 0x18
0xb73b59 8d45e8             LEA EAX, [EBP-0x18]
0xb73b5c 50                 PUSH EAX
0xb73b5d 33f6               XOR ESI, ESI

**************************************************************************
Hook mode: Usermode
Hook type: Inline/Trampoline
Process: 856 (svchost.exe)
Victim module: ntdll.dll (0x7c900000 - 0x7c9b0000)
Function: ntdll.dll!ZwCreateThread 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
0xb73b4d 53                 PUSH EBX
0xb73b4e 56                 PUSH ESI
0xb73b4f 57                 PUSH EDI
0xb73b50 8b7d14             MOV EDI, [EBP+0x14]
0xb73b53 8d4514             LEA EAX, [EBP+0x14]
0xb73b56 50                 PUSH EAX
0xb73b57 6a18               PUSH 0x18
0xb73b59 8d45e8             LEA EAX, [EBP-0x18]
0xb73b5c 50                 PUSH EAX
0xb73b5d 33f6               XOR ESI, ESI
```


2.8.2 `malfind` plugin

One of the most interesting plugins for malware search in memory dumps is `malfind`. This plugin looks for hidden or injected binary code in the user memory space, and depending on the properties of the VAD that contains that memory page and the permissions, it shows it (specifically, pages with write and execution permissions).

This plugin does not detect code injection by forced loading of DLLs (via `CreateRemoteThread` and `LoadLibrary`, since this type of injection can be found through other methods (such as through the `dlldump` plugin, which was discussed before). The `malfind` plugin also admits a parameter `-D`, where you can specify a directory in which to save all the memory pages that it detects as suspicious:

```
[12:01:12] ricardo:volatility git:(master) $ python2 vol.py -f ~/volcados/zeus.vmem --profile=WinXPSP2x86 malfind -p 856 -D /tmp/dumps
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
```

```
Process: svchost.exe Pid: 856 Address: 0xcb0000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 1, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x0000000000cb0000  b8 35 00 00 00 e9 cd d7 c5 7b 00 00 00 00 00 00   .5.......{......
0x0000000000cb0010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x0000000000cb0020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x0000000000cb0030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................

0x0000000000cb0000 b835000000      MOV EAX, 0x35
0x0000000000cb0005 e9cdd7c57b      JMP 0x7c90d7d7
0x0000000000cb000a 0000            ADD [EAX], AL
0x0000000000cb000c 0000            ADD [EAX], AL
0x0000000000cb000e 0000            ADD [EAX], AL
0x0000000000cb0010 0000            ADD [EAX], AL
0x0000000000cb0012 0000            ADD [EAX], AL
0x0000000000cb0014 0000            ADD [EAX], AL
0x0000000000cb0016 0000            ADD [EAX], AL
0x0000000000cb0018 0000            ADD [EAX], AL
```

For each suspicious page, its metadata , a dump of the first bytes and its interpretation in assembly code are shown. Using the native Linux command `file` you can check what has been extracted. In this example it can be seen that two possible executable files have been extracted:

```
[12:09:57] ricardo:volatility git:(master) $ ll /tmp/dumps
total 312
-rw-r--r--  1 ricardo  wheel   152K Jul 13 12:01 process.0x80ff88d8.0xb70000.dmp
-rw-r--r--  1 ricardo  wheel   4.0K Jul 13 12:01 process.0x80ff88d8.0xcb0000.dmp
[12:09:59] ricardo:volatility git:(master) $ file /tmp/dumps/process.*
/tmp/dumps/process.0x80ff88d8.0xb70000.dmp: PE32 executable (GUI) Intel 80386, for MS Windows
/tmp/dumps/process.0x80ff88d8.0xcb0000.dmp: COM executable for DOS
```


2.8.3 `yarascan` plugin

Another plugin of interest is `yarascan`, which allows you to analyze a memory dump through YARA rules. This plugin supports the `--yara-file` parameter, where you can specify a file with the YARA rules you want to apply. Additionally, it allows searching for a simple string, byte patterns, or regular expressions via the `--yara-rules` parameter (for example, `--yara-rules="{eb 90 ff e4 88 32 0d }"` or `--yara-rules="/my(regular|expression{0,1})/"`. By default, the search is performed in user space. If you want to analyze the kernel memory, you have to add the parameter `-K` to the invocation command.

Note that this plugin requires a single file `.yar` against which to parse the contents of the dump. However, YARA rules are usually specified in separate files, being separated by malware families or specific malware samples. In this regard, you may find useful the Python script provided by Andrea Fortuna[1]. This script will allow you, after downloading the content from the official YARA repository (`https://github.com/Yara-Rules`), to obtain a single file `.yar` with all the rules in a single file.

2.8.4 `svcscan` plugin

The `svcscan` plugin allows you to find out which services are registered in the memory dump. As a result of the command, information is obtained about the process of each service (whether it is active or not and whether it belongs to a user process), the original name and the name of the service that is displayed, as well as the type of the service and the actual state. The executable file related to each service is also shown:

```
[12:41:20] ricardo:volatility git:(master) $ python2 vol.py -f ~/volcados/zeus.vmem --profile=WinXPSP2x86 svcscan
Volatility Foundation Volatility Framework 2.6.1
Offset: 0x6e1e90
Order: 1
Start: SERVICE_DISABLED
Process ID: -
Service Name: Abiosdsk
Display Name: Abiosdsk
Service Type: SERVICE_KERNEL_DRIVER
Service State: SERVICE_STOPPED
Binary Path: -

Offset: 0x6e1f20
Order: 2
Start: SERVICE_DISABLED
Process ID: -
Service Name: abp480n5
Display Name: abp480n5
Service Type: SERVICE_KERNEL_DRIVER
Service State: SERVICE_STOPPED
Binary Path: -

Offset: 0x6e1fb0
Order: 3
Start: SERVICE_BOOT_START
Process ID: -
Service Name: ACPI
Display Name: Microsoft ACPI Driver
Service Type: SERVICE_KERNEL_DRIVER
Service State: SERVICE_RUNNING
Binary Path: \Driver\ACPI
```

One parameter of interest for this plugin to detect malicious software that installs itself using `svchost.exe` and implements the actual malicious code in a library of DLL functions is

[1] Available at `https://gist.githubusercontent.com/andreafortuna/29c6ea48adf3d45a979a78763cdc7ce9/raw/4ec711d37f1b428b63bed1f786b26a0654aa2f31/malware_yara_rules.py`

`--verbose`. With this parameter, the registry key `ServiceDLL` is verified and the library associated with said service is reported:

```
[12:54:57] ricardo:volatility git:(master) $ python2 vol.py -f ~/volcados/zeus.vmem --profile=WinXPSP2x86 svcscan --verbose
Volatility Foundation Volatility Framework 2.6.1
Offset: 0x6e1e90
Order: 1
Start: SERVICE_DISABLED
Process ID: -
Service Name: Abiosdsk
Display Name: Abiosdsk
Service Type: SERVICE_KERNEL_DRIVER
Service State: SERVICE_STOPPED
Binary Path: -
ServiceDll:
ImagePath:
FailureCommand:

Offset: 0x6e1f20
Order: 2
Start: SERVICE_DISABLED
Process ID: -
Service Name: abp480n5
Display Name: abp480n5
Service Type: SERVICE_KERNEL_DRIVER
Service State: SERVICE_STOPPED
Binary Path: -
ServiceDll:
ImagePath:
FailureCommand:
```

2.8.5 `ldrmodules` plugin

The `ldrmodules` plugin can be useful for detecting hidden DLLs. In the event that a module were to unbind itself from a process's list of modules, the internal VAD structure that identifies its base address and full disk file path would still exist. With this plugin, a cross-reference is made for each executable file mapped in memory, observing whether or not each module exists in each of the lists of each process.

```
[13:03:49] ricardo:volatility git:(master) $ python2 vol.py -f ~/volcados/zeus.vmem --profile=WinXPSP2x86 ldrmodule
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
     632 winlogon.exe       0x77e70000 True   True   True  \WINDOWS\system32\rpcrt4.dll
     632 winlogon.exe       0x77a80000 True   True   True  \WINDOWS\system32\crypt32.dll
     632 winlogon.exe       0x77fe0000 True   True   True  \WINDOWS\system32\secur32.dll
     632 winlogon.exe       0x5d090000 True   True   True  \WINDOWS\system32\comctl32.dll
     632 winlogon.exe       0x77f60000 True   True   True  \WINDOWS\system32\shlwapi.dll
     632 winlogon.exe       0x771b0000 True   True   True  \WINDOWS\system32\wininet.dll
     632 winlogon.exe       0x77f10000 True   True   True  \WINDOWS\system32\gdi32.dll
```

2.8.6 `idt` plugin

This plugin allows printing the IDT table (*Interrupt Descriptor Table*) of the system for each one of the processors of the machine. It allows to obtain which CPU it refers to, the GDT selector number, the current address and to whom it belongs (which driver and in which section of the

executable it is located). Additionally, it supports a `-v` (or `--verbose`) parameter to display more information about each IDT function.

```
[13:28:53] ricardo:volatility git:(master) $ python2 vol.py -f ~/volcados/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
```

This plugin is useful for finding possible rootkits that modify the IDT entry for `KiSystemService`, taking it to a different kernel module.

2.8.7 `gdt` plugin

This plugin is similar to the previous one, but to query the GDT table (*Global Descriptor Table*) of the system. It allows detection of certain rootkits that install a call gate so that user programs can directly call the system kernel using a `CALL FAR` instruction.

2.8.8 `callbacks` plugin

With this plugin you can detect the notification routines and callbacks to the kernel that are registered in the system from which the memory dump was made. These types of routines are used by rootkits, antivirus, or other dynamic analysis tools and by the Windows operating system itself to monitor and/or react to different events. Specifically, the following can be detected: `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).

```
[12:50:09] ricardo:volatility git:(master) $ python2 vol.py -f ~/volcados/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             -
```


2.8.9 `driverirp` plugin

This plugin is used to print the IRP table of a controller. It locates drivers similar to the `driverscan` plugin, which is another way of looking for kernel drivers in a memory dump. For each controller, it scrolls through the table of functions, printing each function's purpose, address, and the module that owns the address. This command also checks for hooks in IRP functions. Optionally, it prints a disassembly of the first statements in the IRP address with the `-v` option (or `--verbose`).

2.8.10 `devicetree` plugin

This plugin allows knowing the relationship of a controller object with its devices and any other connected device. It is useful to detect possible rootkits in the system.

2.8.11 `timers` plugin

This plugin removes the timers that are installed at the kernel level of the system and the associated DPCs (*Deferred procedure calls*). Some rootkits often make use of DPCs to register timers and launch their activity. This type of behavior can be discovered by looking at possible DPCs that target unknown kernel memory regions.

```
[13:33:38] ricardo:volatility git:(master) $ python2 vol.py -f ~/volcados/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
```

2.8.12 `getsids` plugin

The `getsids` plugin allows you to view the security identifiers (*Security Identifiers*, SIDs) associated with a process. This information is useful for identifying processes that have escalated privileges and for verifying which user each process belongs to.


```
[13:54:19] ricardo:volatility git:(master) $ python2 vol.py -f ~/volcados/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)
services.exe (676): S-1-5-18 (Local System)
services.exe (676): S-1-5-32-544 (Administrators)
services.exe (676): S-1-1-0 (Everyone)
services.exe (676): S-1-5-11 (Authenticated Users)
```

2.8.13 `privs` plugin

The `privs` plugin allows you to obtain information about the privilege tokens that each process has. This plugin supports the `--silent` parameter to show only those privileges that a process has enabled (those that are not active by default, but are later enabled). It can also be used with the `--regex` parameter to filter on a specific privilege name.

```
[13:56:01] ricardo:volatility git:(master) $ python2 vol.py -f ~/volcados/zeus.vmem --profile=WinXPSP2x86 privs
Volatility Foundation Volatility Framework 2.6.1
Pid      Process          Value  Privilege                         Attributes              Description
-------- ---------------- ------ --------------------------------- ----------------------- -----------
       4 System                7 SeTcbPrivilege                    Present,Enabled,Default  Act as part of the operating system
       4 System                2 SeCreateTokenPrivilege            Present                  Create a token object
       4 System                9 SeTakeOwnershipPrivilege          Present                  Take ownership of files/objects
       4 System               15 SeCreatePagefilePrivilege         Present,Enabled,Default  Create a pagefile
       4 System                4 SeLockMemoryPrivilege             Present,Enabled,Default  Lock pages in memory
       4 System                3 SeAssignPrimaryTokenPrivilege     Present                  Replace a process-level token
       4 System                5 SeIncreaseQuotaPrivilege          Present                  Increase quotas
       4 System               14 SeIncreaseBasePriorityPrivilege   Present,Enabled,Default  Increase scheduling priority
       4 System               16 SeCreatePermanentPrivilege        Present,Enabled,Default  Create permanent shared objects
       4 System               20 SeDebugPrivilege                  Present,Enabled,Default  Debug programs
       4 System               21 SeAuditPrivilege                  Present,Enabled,Default  Generate security audits
       4 System                8 SeSecurityPrivilege               Present                  Manage auditing and security log
       4 System               22 SeSystemEnvironmentPrivilege      Present                  Edit firmware environment values
       4 System               23 SeChangeNotifyPrivilege           Present,Enabled,Default  Receive notifications of changes to files or directories
       4 System               17 SeBackupPrivilege                 Present                  Backup files and directories
       4 System               18 SeRestorePrivilege                Present                  Restore files and directories
       4 System               19 SeShutdownPrivilege               Present                  Shut down the system
       4 System               10 SeLoadDriverPrivilege             Present                  Load and unload device drivers
       4 System               13 SeProfileSingleProcessPrivilege   Present,Enabled,Default  Profile a single process
       4 System               12 SeSystemtimePrivilege             Present                  Change the system time
       4 System               25 SeUndockPrivilege                 Present                  Remove computer from docking station
       4 System               28 SeManageVolumePrivilege           Present                  Manage the files on a volume
       4 System               29 SeImpersonatePrivilege            Present,Enabled,Default  Impersonate a client after authentication
       4 System               30 SeCreateGlobalPrivilege           Present,Enabled,Default  Create global objects
     544 smss.exe              7 SeTcbPrivilege                    Present,Enabled,Default  Act as part of the operating system
     544 smss.exe              2 SeCreateTokenPrivilege            Present                  Create a token object
     544 smss.exe              9 SeTakeOwnershipPrivilege          Present                  Take ownership of files/objects
```

2.8.14 `verinfo` plugin

This plugin allows to obtain the information of the executable files of the processes contained in the dump (only of user space and both of the executable modules and of the DLL libraries). Note that the information provided by this plugin is not reliable, since not all files contain this metadata and because the information in malicious code files is often false. Additionally, the `--regex=REGEX` and `--ignore-case` parameters can be used to filter on a given name.


```
[13:56:07] ricardo:volatility git:(master) $ python2 vol.py -f ~/volcados/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
```

2.8.15 `envars` plugin

This plugin allows you to view the environment variables for each process. It shows, among other things, the number of installed processors, the variable `Path`, the current directory of the process, the temporary directory, the machine name, user name, etc.

```
[14:10:39] ricardo:volatility git:(master) $ python2 vol.py -f ~/volcados/zeus.vmem --profile=WinXPSP2x86 envars
Volatility Foundation Volatility Framework 2.6.1
Pid      Process            Block      Variable                       Value
-------- ------------------ ---------- ------------------------------ -----
     608 csrss.exe          0x00100000 ComSpec                        C:\WINDOWS\system32\cmd.exe
     608 csrss.exe          0x00100000 FP_NO_HOST_CHECK               NO
     608 csrss.exe          0x00100000 NUMBER_OF_PROCESSORS           1
     608 csrss.exe          0x00100000 OS                             Windows_NT
     608 csrss.exe          0x00100000 Path                           C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem
     608 csrss.exe          0x00100000 PATHEXT                        .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH
     608 csrss.exe          0x00100000 PROCESSOR_ARCHITECTURE         x86
     608 csrss.exe          0x00100000 PROCESSOR_IDENTIFIER           x86 Family 6 Model 23 Stepping 10, GenuineIntel
     608 csrss.exe          0x00100000 PROCESSOR_LEVEL                6
     608 csrss.exe          0x00100000 PROCESSOR_REVISION             170a
     608 csrss.exe          0x00100000 SystemDrive                    C:
     608 csrss.exe          0x00100000 SystemRoot                     C:\WINDOWS
     608 csrss.exe          0x00100000 TEMP                           C:\WINDOWS\TEMP
     608 csrss.exe          0x00100000 TMP                            C:\WINDOWS\TEMP
     608 csrss.exe          0x00100000 windir                         C:\WINDOWS
     632 winlogon.exe       0x00010000 ALLUSERSPROFILE                C:\Documents and Settings\All Users
     632 winlogon.exe       0x00010000 APPDATA                        C:\Documents and Settings\Administrator\Application Data
     632 winlogon.exe       0x00010000 CommonProgramFiles             C:\Program Files\Common Files
     632 winlogon.exe       0x00010000 COMPUTERNAME                   BILLY-DB5B96DD3
     632 winlogon.exe       0x00010000 ComSpec                        C:\WINDOWS\system32\cmd.exe
```

