**Memory Forensics**

**In this experiment, analyze the RAM memory dump using volatility tool from publicly available memory samples** [**https://code.google.com/archive/p/volatility/wikis/FAQ.wiki**](https://code.google.com/archive/p/volatility/wikis/FAQ.wiki)

**Here, be2.vmem available in the soft could be used**

**Digging through memory can be an effective way to identify indicators of compromise.  Malware that leverages root kit techniques can fool many tools that run within the OS.  Malware authors have ways of hiding their malicious code from various Windows data structures which can help them avoid detection.  A trivial example of this is when malware removes itself from the \_EPROCESS doubly linked-list, this causes the process to not show up in Task Manager, but since the process still has threads in memory it will still run.**

**By analyzing memory you can detect the following:**

**– Running Processes**

**– Passwords/Encryption Keys**

**– Unpacked/decrypted PEs**

**– Registry Hives**

**– File/web page artifacts**

**– Previously typed commands**

**Creating a Raw Memory Dump:**

**First a raw memory image must be created from the system. This involves taking what is running in RAM and saving it to a file called a memory dump. In Windows, memory is managed in both physical RAM and virtual memory through the use of a paging file. It’s important to note that you will only be dumping memory from physical memory (RAM). It is possible that some of the interesting memory segments will be in virtual memory (page file) and won’t be captured in your physical memory dump.**

**There are several methods you can leverage for memory dump acquisition:  
– Monsools Win32dd/Win64dd  
– FTK Imager  
– VMware snapshots (.vmem file) = raw memory image. VirtualBox (.sav file) = partial image  
– Cuckoo Sandbox can be leveraged to automatically get a memory dump  
– memdump for Linux systems**

**Acquiring memory dump from RAM is not covered. Try using memdump to acquire memory dump of RAM and report it as part of the experiment.**

**Memory Analysis using Volatility**

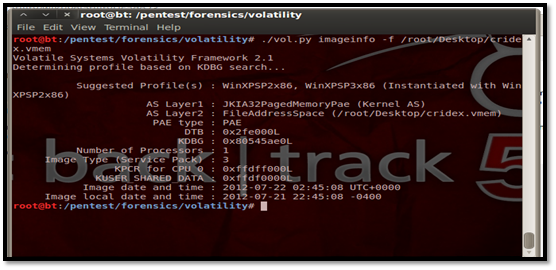
**Traverse to the path that contains vol.py and run the following command**

**python vol.py <command> -f <full path of the memory dump file>**

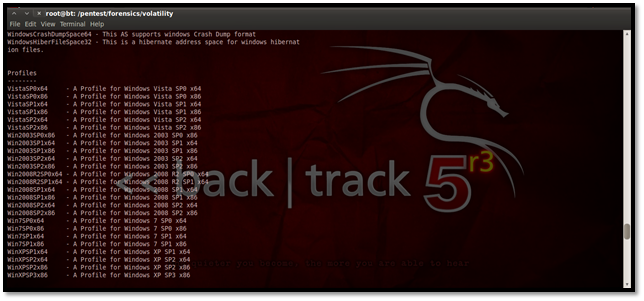
**Selecting a Profile**

For performing analysis using Volatility , It is necessary to first set a profile to tell Volatility what operating system the dump came from, such as Windows XP, Vista, Linux flavors, etc.

Imageinfo plug-in could be used to find this out.



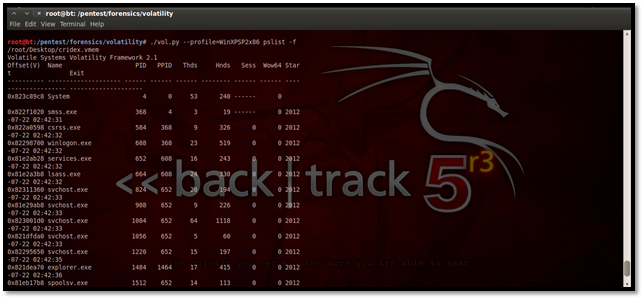
From the above screenshot, Volatility suggests using the profile for Windows XP SP2 x86 or Windows XP SP3 x86. Let us select Windows SP2 x86. The default profile for Volatility is WinXPSP2x86 if profile is not set explicitly.

Here is the list of the available profiles in Volatility. 

**Viewing Running Processes**

This plug-in gives us the option to view all running process on the particular system during which the memory dump was taken.

**python vol.py –profile=WinXPSP2x86 pslist –f <filename>**



The above screenshot shows a clear view of all the processes running during the memory dump.

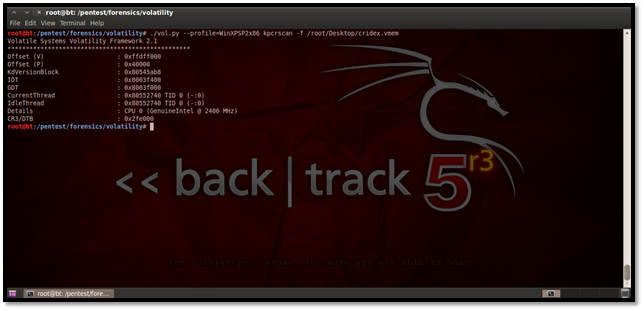
**Using Kdbgscan**

This particular plug-in is designed to positively identify the correct profile of the system and the correct KDBG (kernel debugger block) address. It simply scans for KDBG header signatures linked to the profiles in Volatility. This is mainly helpful in clearing up confusions which might be caused if the Pslist plug-in not showing any processes in the process list. This may happen if a KDBG with an invalid PsActiveProcessHead pointer is found earlier in a sample.



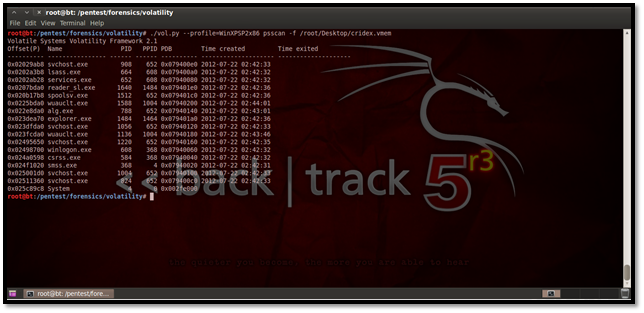
**Using Kpcrscan**

This plug-in is used to scan for KPCR (Kernel Processor Control Region) structures. A KPCR is a data structure used by the kernel to store the processor-specific data. Kpcrscan searches for and dumps potential KPCR values. Each processor on a multi-core system has its own KPCR. In the screenshot below we can see the details of the processor, which is a single-core processor.



Using Psscan for Malware Analysis

This plug-in is mostly used for malware analysis and scanning rootkit activities. It scans for inactive, hidden and unlinked processes by a rootkit/malware. Here’s how we do it:

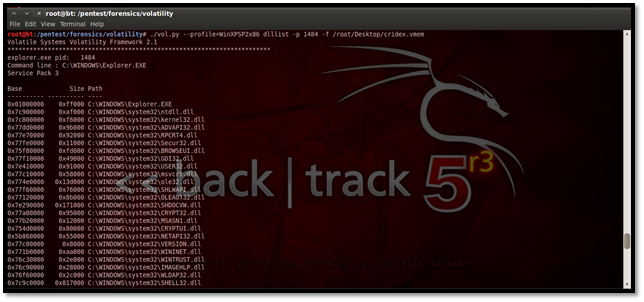


**Using Dlllist**

To display the DLLs for all currently running processes or a particular process we use this plug-in.



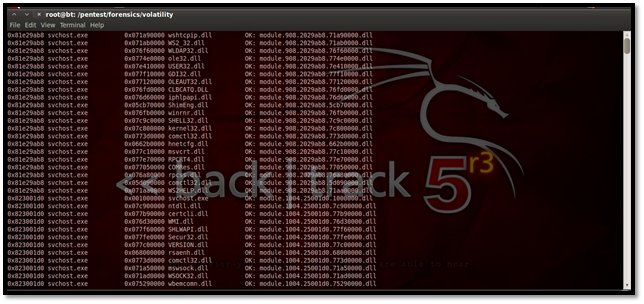
For listing the DLLs for a specific process, suppose we list here the DLLs of explorer.exe, which has the process id 1484. The process id may be found using the pslist plug-in.



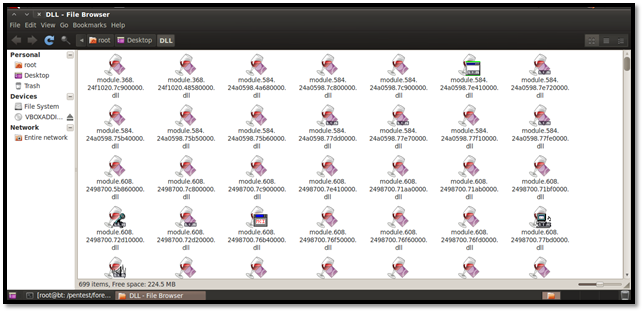
**Using Dlldump**

We can dump all the DLLs for further forensic analysis using the command:

./vol.py –profile=WinXPSP2x86 dlldump -D <Destination Directory> -f <memory image location>

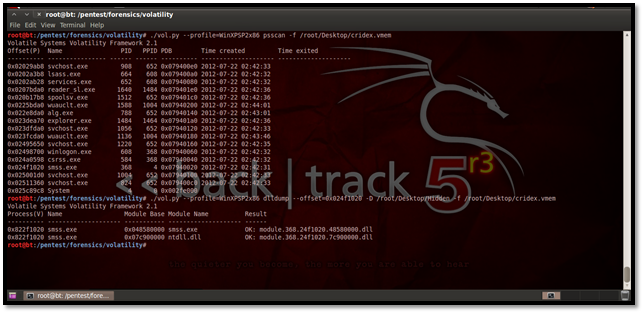


We can see the dump of the DLLs in the directory below:

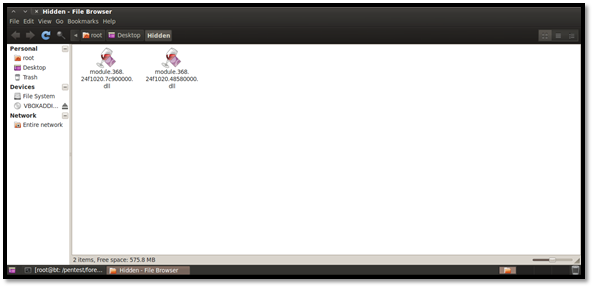


We can even dump DLLs from specific processes if we figure out that a malicious process may have been running. Similarly, we can dump DLLs of a hidden process by using its offset address as shown below.

Here is a list of all hidden processes once again. Now we have used the offset address for smss.exe, which is 0x024f1020 and dumped the DLLs in the folder named Hidden.

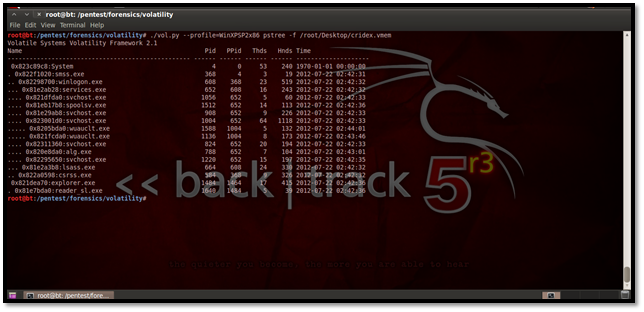


Here are the DLLs from the hidden process smss.exe:



**Using Pstree**

This plug-in may be used for viewing the processes in a tree form. Similar to Pslist, it does not show the hidden processes. Here is how it looks:



**Using Consoles to Find Commands Used in Cmd.exe**

This plug-in is used to find the various commands typed in locally or remotely via backdoors. Since our capture did not have any typed in commands in cmd.exe, we get no results here:



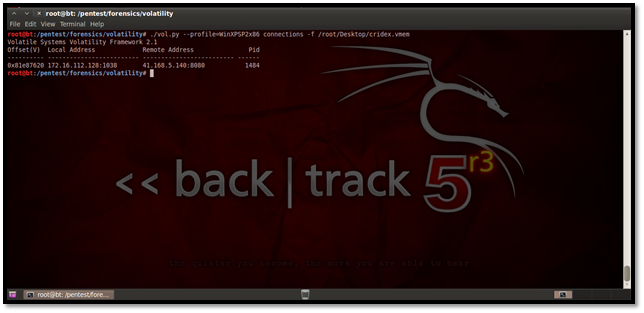
**Using Verinfo for Displaying Version Information**

This plug-in helps to display the version information for portable executable files.



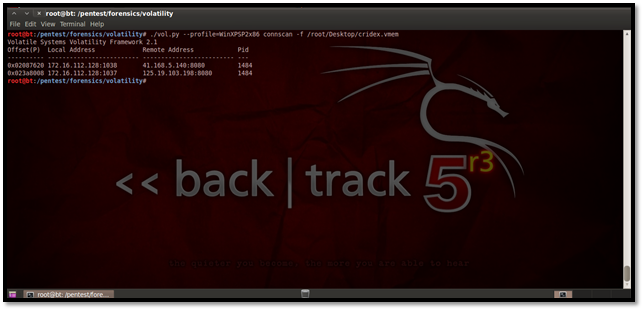
From the above screenshot we can see the file version, product version, OS, file type, etc.  
**Using Connections Plug-In to Display TCP Connections**

This plug-in is used to display TCP connections that were active at the time the memory dump was taken. You can see this in the below screenshot:



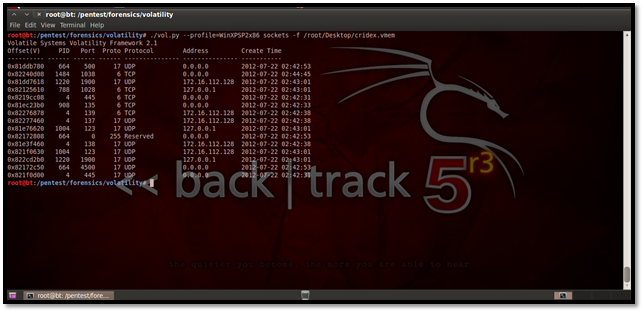
**Using Connscan**

The connscan plug-in helps us to find active connections as well as connections that might have been terminated. Here is how to use it:



**Using Sockets**

This plug-in helps us to find out the listening socket connections during the time our memory dump was taken. These include TCP as well as UDP connections.



**Using Hivescan**

This plug-in helps us to find physical addresses of registry hives in memory.



**Using Hivelist**

This plug-in helps us to find the virtual addresses of registry hives in memory.



**Using Svcscan**

This plug-in helps us to find the list of services running on the system.

