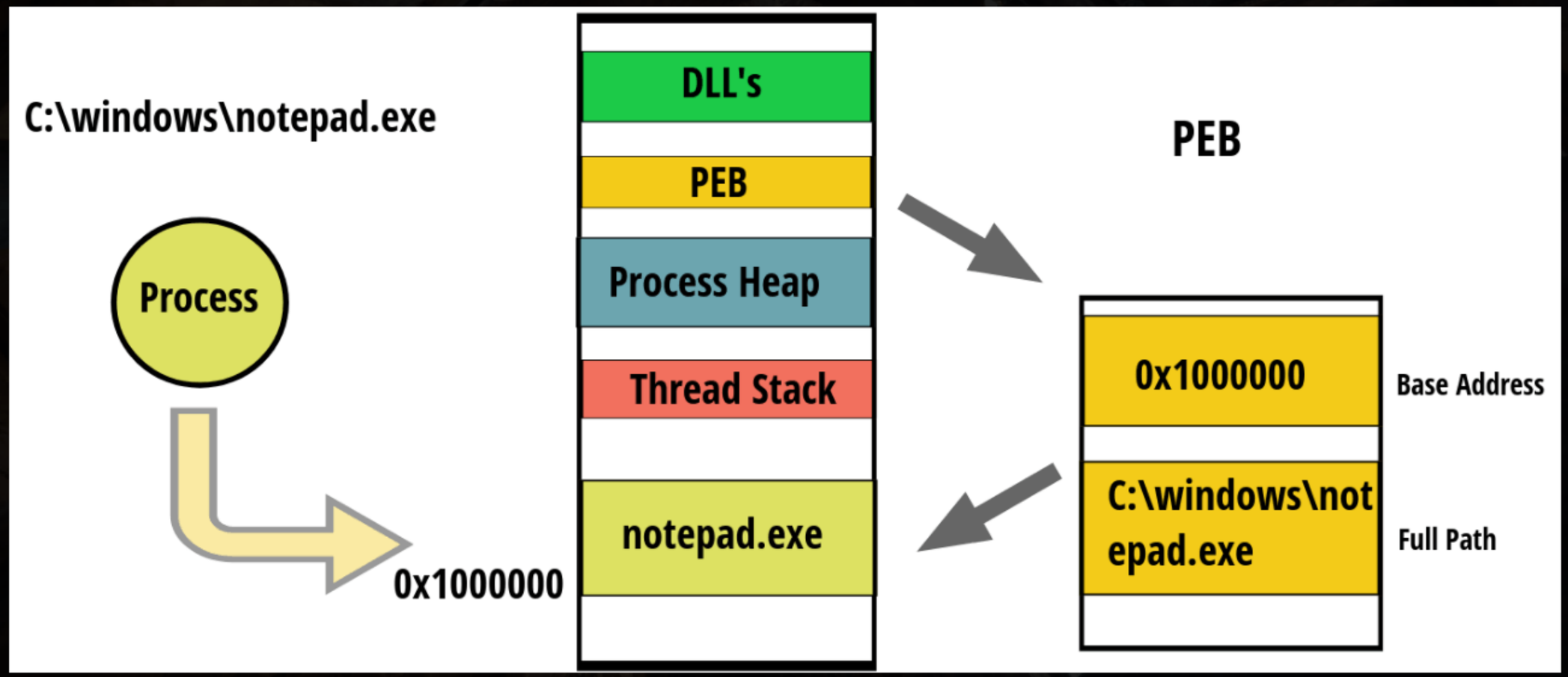
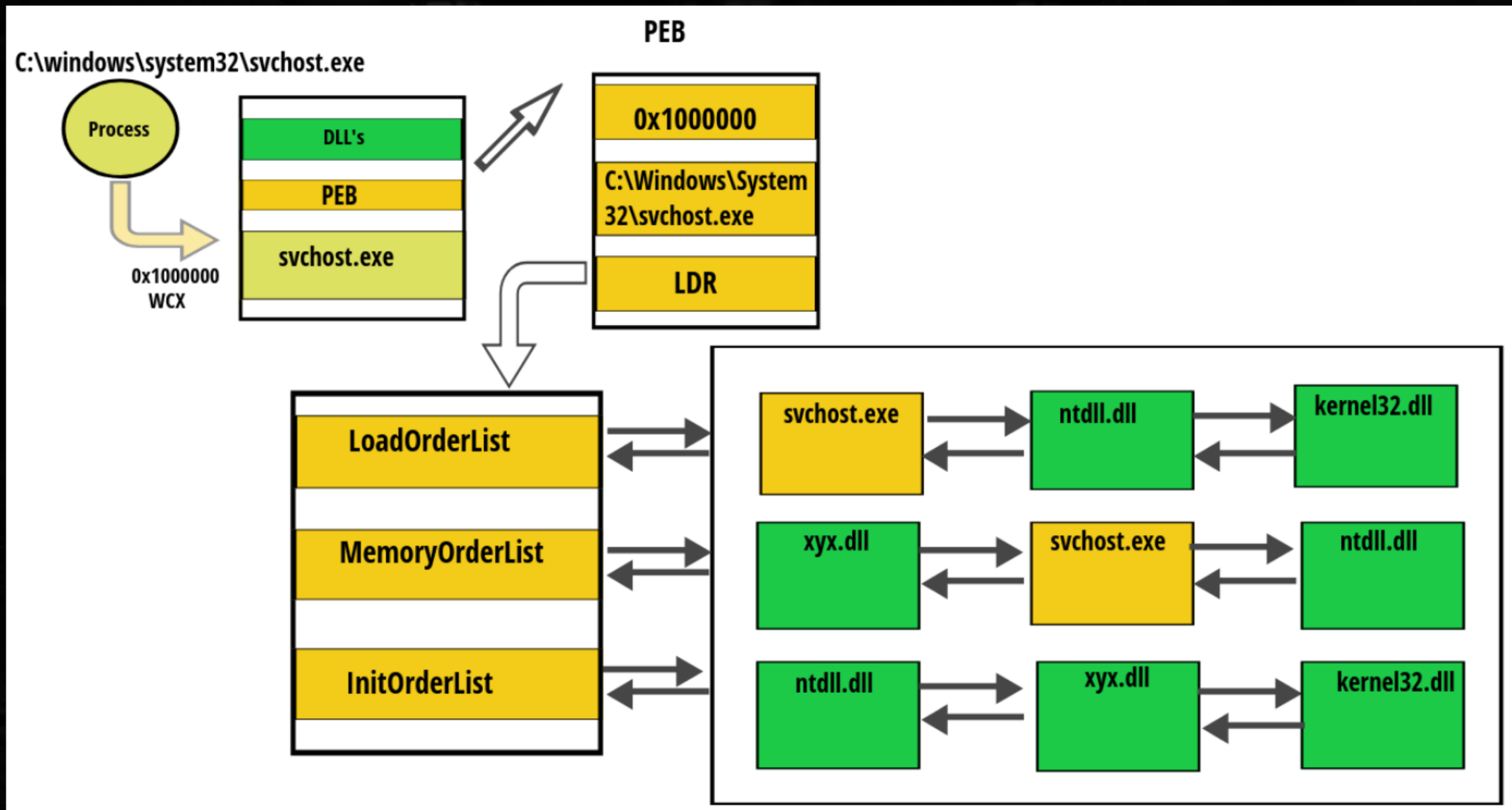
Investigating Process Memory

Process Memory

- Each process thinks it has flat linear memory
- Process memory ranges from 0 to MmHighestUserAddress (symbol in the NT module)
- Internally, virtual memory is mapped to physical memory, but may also be stored on disk
- The memory manager handles mapping of virtual to physical pages
- Process Memory consists of DLL's, Environment Variables, PEB, Heaps, Stacks, Mapped Files and Executable

Process Memory




PEB
C:\windows\system32\svchost.exe
Process
DLL's
PEB
svchost.exe
0x1000000
WCX
0x1000000
C:\Windows\System 32\svchost.exe
LDR
LoadOrderList
MemoryOrderList
InitOrderList
svchost.exe
ntdll.dll
kernel32.dll
xyx.dll
svchost.exe
ntdll.dll
ntdll.dll
xyx.dll
kernel32.dll

Listing DLL's using ***dlllist*** plugin

dlllist lists the modules by walking the double-linked of ***InLoadOrderModuleList***

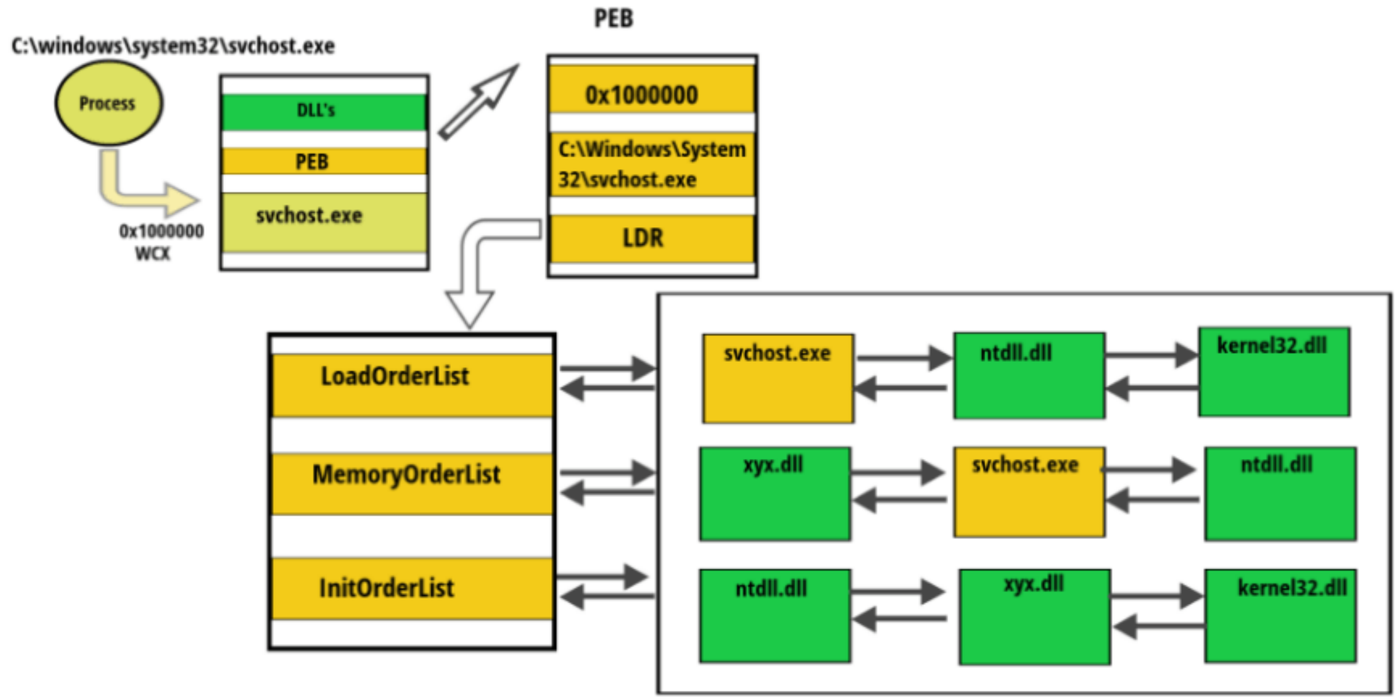
```
root@kratos:~/Volatility# python vol.py -f rat9002.vmem dlllist -p 1456
Volatility Foundation Volatility Framework 2.5
************************************************************************
rundll32.exe pid:   1456
Command line : rundll32.exe c:\ws2_32.dll,runeree
Service Pack 3

Base             Size  LoadCount Path
---------- ---------- ---------- ----
0x01000000     0xb000     0xffff C:\WINDOWS\system32\rundll32.exe
0x7c900000    0xaf000     0xffff C:\WINDOWS\system32\ntdll.dll
0x7c800000    0xf6000     0xffff C:\WINDOWS\system32\kernel32.dll
0x77c10000    0x58000     0xffff C:\WINDOWS\system32\msvcrt.dll
0x77f10000    0x49000     0xffff C:\WINDOWS\system32\GDI32.dll
0x7e410000    0x91000     0xffff C:\WINDOWS\system32\USER32.dll
0x76c90000    0x28000     0xffff C:\WINDOWS\system32\IMAGEHLP.dll
0x5cb70000    0x26000        0x1 C:\WINDOWS\system32\ShimEng.dll
0x6f880000   0x1ca000        0x1 C:\WINDOWS\AppPatch\AcGenral.DLL
0x77dd0000    0x9b000       0x6d C:\WINDOWS\system32\ADVAPI32.dll
0x77e70000    0x92000       0x2f C:\WINDOWS\system32\RPCRT4.dll
```

Listing Modules using ldrmodules Plugin

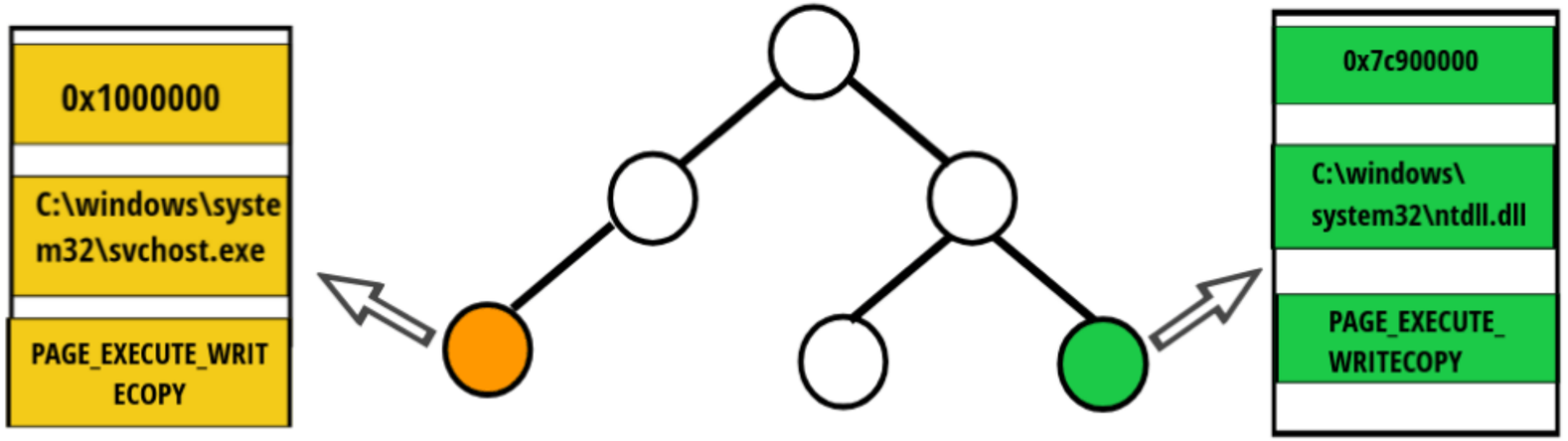
ldrmodules plugin compares the PEB lists with data in the VAD

```
root@kratos:~/Volatility# python vol.py -f rat9002.vmem ldrmodules -p 1456
Volatility Foundation Volatility Framework 2.5
Pid      Process              Base       InLoad InInit InMem MappedPath
-------- -------------------- ---------- ------ ------ ----- ----------
1456 rundll32.exe         0x01000000 True   False  True  \WINDOWS\system32\rundll32.exe
1456 rundll32.exe         0x00a10000 True   True   True  \WINDOWS\system32\normaliz.dll
1456 rundll32.exe         0x76b40000 True   True   True  \WINDOWS\system32\winmm.dll
1456 rundll32.exe         0x769c0000 True   True   True  \WINDOWS\system32\userenv.dll
1456 rundll32.exe         0x76f60000 True   True   True  \WINDOWS\system32\wldap32.dll
1456 rundll32.exe         0x77c00000 True   True   True  \WINDOWS\system32\version.dll
1456 rundll32.exe         0x5ad70000 True   True   True  \WINDOWS\system32\uxtheme.dll
1456 rundll32.exe         0x68000000 True   True   True  \WINDOWS\system32\rsaenh.dll
1456 rundll32.exe         0x76fc0000 True   True   True  \WINDOWS\system32\rasadhlp.dll
1456 rundll32.exe         0x71ab0000 True   True   True  \WINDOWS\system32\ws2_32.dll
1456 rundll32.exe         0x73dd0000 True   True   True  \WINDOWS\system32\mfc42.dll
1456 rundll32.exe         0x77be0000 True   True   True  \WINDOWS\system32\msacm32.dll
1456 rundll32.exe         0x10000000 True   True   True  \ws2_32.dll
```



User Memory

Kernel Memory



Dumping an executable from Memory

Procdump plugin can be used to dump the executable from memory to disk, with ***-p*** option you can specify the PID to dump and with ***-D*** option you can specify the directory where the process executable will be dumped

```
root@kratos:~/Volatility# python vol.py -f rat9002.vmem procdump -p 1456 -D dump/
Volatility Foundation Volatility Framework 2.5
Process(V) ImageBase  Name                 Result
---------- ---------- -------------------- ------
0x8152bda0 0x01000000 rundll32.exe         OK: executable.1456.exe
root@kratos:~/Volatility#
```

Dumping DLL from Memory

The ***dlldump*** plugin can be used to dump the DLL from memory to disk, with ***-p*** option you can specify the process, with ***-b*** option you can specify the base address of the DLL and with ***-D*** option you can specify the directory to dump the DLL.

```
root@kratos:~/Volatility# python vol.py -f rat9002.vmem dlldump -p 1456 -b 0x10000000
-D dump/
Volatility Foundation Volatility Framework 2.5
Process(V) Name                 Module Base Module Name          Result
---------- -------------------- ----------- -------------------- ------
0x8152bda0 rundll32.exe         0x010000000 ws2_32.dll           OK: module.1456.172bd
a0.10000000.dll
root@kratos:~/Volatility#
```

Scanning Patterns using *yarascan* plugin

Yarascan can be used to scan for ***strings***, a ***sequence of bytes*** or ***regex*** in process or kernel memory

```
root@kratos:~/Volatility# python vol.py -f rat9002.vmem yarascan -Y "MZ"
Volatility Foundation Volatility Framework 2.5
Rule: r1
Owner: Process System Pid 4
0x7c900000  4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00   MZ..............
0x7c900010  b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00   ........@.......
0x7c900020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x7c900030  00 00 00 00 00 00 00 00 00 00 00 00 e0 00 00 00   ................
0x7c900040  0e 1f ba 0e 00 b4 09 cd 21 b8 01 4c cd 21 54 68   ........!..L.!Th
0x7c900050  69 73 20 70 72 6f 67 72 61 6d 20 63 61 6e 6e 6f   is.program.canno
0x7c900060  74 20 62 65 20 72 75 6e 20 69 6e 20 44 4f 53 20   t.be.run.in.DOS.
0x7c900070  6d 6f 64 65 2e 0d 0d 0a 24 00 00 00 00 00 00 00   mode....$.......
```

```
root@kratos:~/Volatility# python vol.py -f rat9002.vmem yarascan -Y "{4d 5a}"
Volatility Foundation Volatility Framework 2.5
Rule: r1
Owner: Process System Pid 4
0x7c900000  4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00   MZ..............
0x7c900010  b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00   ........@.......
0x7c900020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x7c900030  00 00 00 00 00 00 00 00 00 00 00 00 e0 00 00 00   ................
0x7c900040  0e 1f ba 0e 00 b4 09 cd 21 b8 01 4c cd 21 54 68   ........!..L.!Th
0x7c900050  69 73 20 70 72 6f 67 72 61 6d 20 63 61 6e 6e 6f   is.program.canno
0x7c900060  74 20 62 65 20 72 75 6e 20 69 6e 20 44 4f 53 20   t.be.run.in.DOS.
0x7c900070  6d 6f 64 65 2e 0d 0d 0a 24 00 00 00 00 00 00 00   mode....$.......
```

Lab 13.2: The Case of Sypbot (contd.)

Use the memory image (***spybot.vmem***) :

- Dump the malicious process to the disk?
- Can you confirm if the dumped process is associated with spybot?
- Can you confirm if the malware is using IRC for its communication?

Lab 14 - The Case of Prolaco

While monitoring for security events, you determine that one of the hosts (192.168.1.100) is sending a spam email, you suspect this host to be infected with a spambot, you collect the memory image (prolaco.vmem) from the host. use memory image prolaco.vmem to answer the below questions:

- Does the network connection show any indication of the host sending the spam?
- Can you determine the malicious process id that is responsible for spam activity?
- Can you determine the name of the malicious process that is sending the spam?
- Can you dump the malicious process from the memory?
- Can you confirm, if the dumped process is malicious?
- Can you determine any unique indicator associated with this malware?
- Is there any other process that is related to the malicious process?

Investigating Usermode Rootkits/Fileless malware

Code Injection

- Malware uses code injection to perform actions from within the context of another process

- Malware can force a legitimate process to perform actions on its behalf like stealing information

- This is a means of stealth and persistence

- Attackers can inject code into a process in many ways, such as writing to the remote process memory directly or adding a registry key that makes new processes load a DLL of the attacker's choice.

- Volatility plugins to detect API hooks and code injection: ***apihooks*** and ***malfind***

Different types of Code Injection

- ***Remote Library injection*** - uses api ***LoadLibrary*** or Native ***LdrLoadDll***
- ***Remote Code Injection*** - Writes code into the target process and forces it to execute
- ***Reflective DLL injection*** - A malicious process writes a DLL (as a sequence of bytes) into the memory space of a target process, the DLL then initializes itself without the help of Windows Loader.
- ***Process Hollowing*** - A malicious process starts a new instance of a legitimate process (such as ***lsass.exe***) in suspended mode. Before resuming it, the executable section(s) are freed and reallocated with malicious code.

Demo 14 - Investigation of SpyEye

Lab 15 - The Case of Zegost

Your security device alerts on a malware callback connection from ***192.168.1.60*** to the C2 domain "***xntk0520.9966.org***" on port ***8000*** (as shown in the screenshot), the C2 domain resolves to IP ***192.168.1.22***. You suspect the host ***192.168.1.60*** to be infected. You collect the memory image from the host (***zegost.vmem***).

- Which process is connecting to the C2 server?
- What is the full path of the process?
- Is this a legitimate process?
- If it is a legitimate process then why is the process connecting to the C2 ip and can you identify the component that is malicious and dump it to disk?
- Can you establish any relationship between the dumped component and the C2 domain?

Lab 16 - The Case of Taidoor Espionage

Your security device alerts on a malware callback connection from ***192.168.1.60*** to ***200.2.126.61*** on port ***443***. you suspect the host ***192.168.1.60*** to be infected. you collect the memory image from the host (***taidoor.vmem***). Analyze the memory image and answer the below questions

- Can you confirm if the host made the connection to the C2 server?
- What is the process id of the malicious process?
- Can you determine the full path of the malicious process and based on the path do you think its a legitimate operating system process?
- If this is a legitimate process, is there is anything that makes it different from the legitimate process?
- Dump the process onto disk, can you recognize any interesting strings?
- Based on your observation, what code injection technique malware is using?