Capstone: Volatility and Memory forensics

Daniel Pepin
CSCI 400
Prof. Hassan

Attack Analysis of Volatile Memory, also described as memory forensics, is an integral part of computer security and our ability to thwart potential threats from affecting our systems whether they are personal or work systems. Memory Forensics and the tools created to practice this type of investigation has gone through many generations starting from Zeroth generation tools, to the most recent generation being Third generation tools. The most important use of these tools are to analyze malware that are used in order to gain an assortment of things out of attacking systems owned by normal users or industry systems and their entire complex of networks and computers. Malware whether it is ransomware, a key logger, or most importantly a rootkit, we need all of the various tools created and actively updated to help us gain insight on how these attacks are engineered and how they can be combated as something like a rootkit is able to hide itself from the user and basic memory viewing options provided by the os in use. “Memory forensics is forensic analysis of a computer's memory dump. Its primary application is investigation of advanced computer attacks which are stealthy enough to avoid leaving data on the computer's hard drive. Consequently, the memory (RAM) must be analyzed for forensic information.” (“Memory_forensics” 2020) With our understanding of Memory forensics it’s easy to think of a use case and understand the importance of this practice to help protect us against more advanced malware that your average anti-virus.

With many generations of tools the generation highlighted is the second generation of tools; “Volatility is an open-source memory forensics framework for incident response and malware analysis” (“Volatility (Memory Forensics)” 2020). Volatility 2.6 now heading into beta 3.0 development was created initially by a man by the name, Aaron Walters and is currently maintained and updated actively by the community as a virtue of it being open source. It was built primarily in python and is easily run in Windows, various versions of linux and MacOS High Sierra and older. Volatility takes a given memory profile in order to analyze a memory dump, crash dump, or even memory samples of malware. A prominent memory sample to analyze is a sample of stuxnet. As well as that, Volatility has an extremely important use of api hooks that allow for analysis of rogue DLLs and Drivers. “Memory forensics is a powerful investigation technique and with a tool like Volatility it is possible to find advanced malware and its forensic artifacts from the memory” (Finding Advanced Malware Using Volatility 2020) Volatility has become quite important for the reverse engineering of new and old malware, as well and finding the the many parts of a virus that doesn't just run in one single memory location but injects itself into your drivers and other processes. Being able to track down all of the various parts hidden within your system. An important thing to note is that Volatility also allows for one to see whether or not a process is connecting to any IP addresses allowing you to have even more information on the processes running. If a process were to be running and connecting to a domain like web3inst.com which is a well known domain related to malware and we were to gain this information on the process we can then speed the process of dealing with such malware on our systems.

Within this project I find it to be crucial to analyze and get memory samples and analyze the samples of multiple viruses on a Kali Linux virtual machine.providing a tutorial for volatility framework I hope to be able to make it easier to identify rogue processes, Analyze process DLLs and handles, Review network artifacts, Look for evidence of code injection, Check for signs of rootkit, and Dump suspicious processes and drivers. Using this methodology I hope to gain insight on possible common practices for the development of these malware.

Volatile Memory

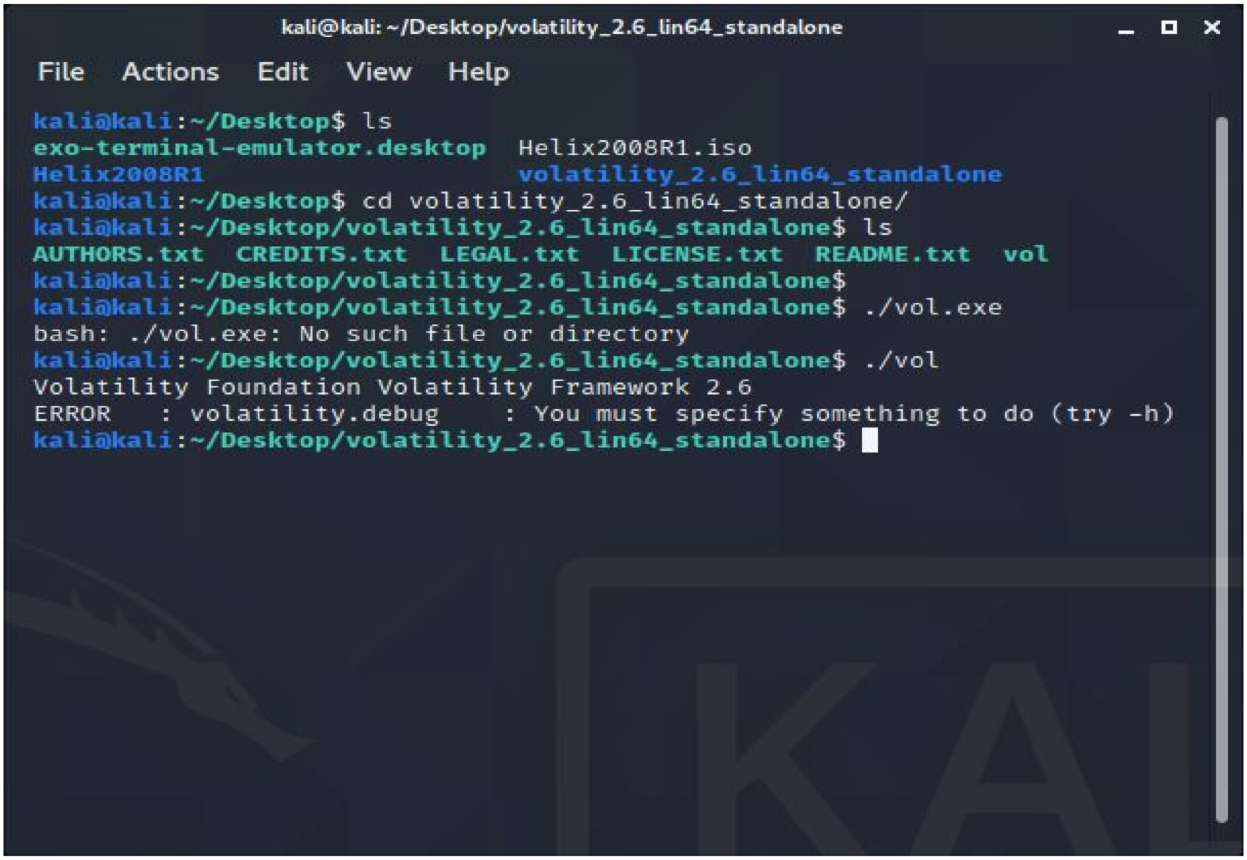
Before using Volatility to analyze volatile memory and memory samples, it is very important to understand what volatile memory is exactly. Volatile memory is the hardware that fetches and stores data at request at high speeds. This memory is temporary as the name volatile memory suggests. RAM (Random Access Memory) Is a hardware component meant to temporarily store fetched data allowing for the cpu to actively process information stored. In contrast, non volatile memory is simply your hard drive, solid state drive or read-only memory such as optical disks, floppy discs and so on. It's important to understand each one's purpose in general but when it comes to the forensics and studying of malware what we need to understand is that different malware impacts these two different parts of a system differently. While many different malware effects and seed themselves in non volatile memory, an interesting problem arises when that malware deletes its own signature from the hard drive and is hosted within memory.

Memory-Resident Malware

Memory-resident malware also known as fileless malware perform their core functions without writing data to disk during the lifetime of their operation allowing for it to become increasingly hard to track and report as traditional anti-malware generally searches through disk in order to find potential malware. “These techniques evolved by way of temporary memory resident viruses and were seen in famous examples such as: Anthrax, Monxla and took on their truer "fileless" nature by way of in-memory injected network viruses/worms such as CodeRed and Slammer. More modern evolutionary incarnations have been seen in viruses such as Stuxnet, Duqu, Poweliks, Phasebot etc" (Wikipedia contributors, 2020) Memory resident malware has become more and more popular as it leaves no trace on the systems hard drive.

Prerequisites for Volatility Framework

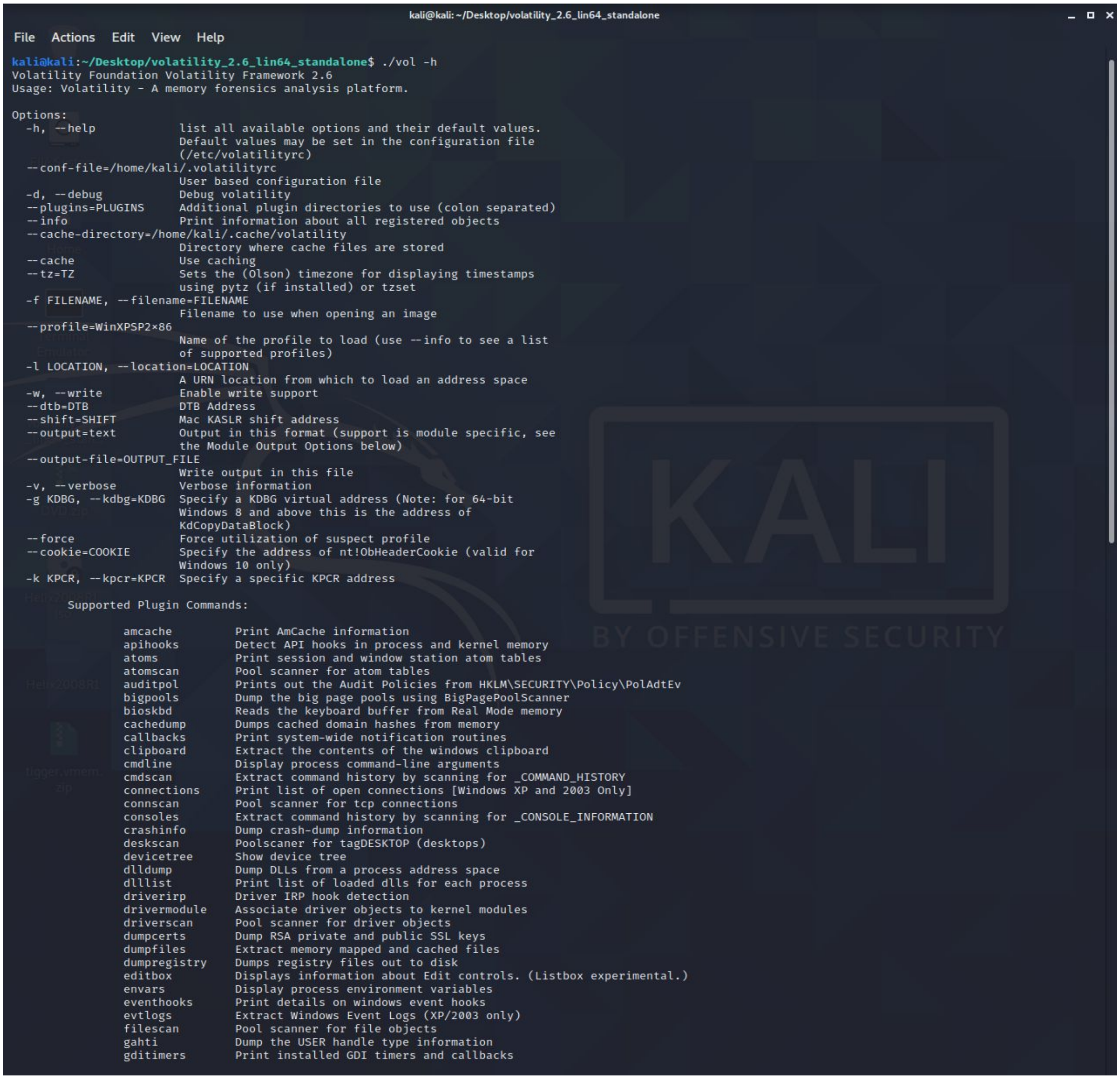
Installation - Installation for Volatility is quite simple. For the purpose of safety of my own system I will be installing volatility on a kali linux virtual machine as well as the memory samples. The download is available at https://www.volatilityfoundation.org/releases and I will be using the Linux Standalone Executables.



```
kali@kali:~/Desktop$ ls
exo-terminal-emulator.desktop  Helix2008R1.iso
Helix2008R1                    volatility_2.6_lin64_standalone
kali@kali:~/Desktop$ cd volatility_2.6_lin64_standalone/
kali@kali:~/Desktop/volatility_2.6_lin64_standalone$ ls
AUTHORS.txt  CREDITS.txt  LEGAL.txt  LICENSE.txt  README.txt  vol
kali@kali:~/Desktop/volatility_2.6_lin64_standalone$
kali@kali:~/Desktop/volatility_2.6_lin64_standalone$ ./vol.exe
bash: ./vol.exe: No such file or directory
kali@kali:~/Desktop/volatility_2.6_lin64_standalone$ ./vol
Volatility Foundation Volatility Framework 2.6
ERROR   : volatility.debug    : You must specify something to do (try -h)
kali@kali:~/Desktop/volatility_2.6_lin64_standalone$ 
```

Upon download I place it on the desktop and in order to access the executable itself all that needs to be done is change directory into the file:///home/kali/Desktop/volatility_2.6_lin64_standalone.

Within the file location there will be an executable named volatility_2.6_win64_standalone.exe. This executable is volatility itself and what you will be interacting with for the most part, and while typing out the name of that executable in the terminal is very repetitive and time consuming thankfully as you can see, we can rename it to whatever we want as I renamed the executable to vol. You will also notice running the executable without some argument passed like -h will return an error for us.


kali@kali: ~/Desktop/volatility_2.6_lin64_standalone
File Actions Edit View Help

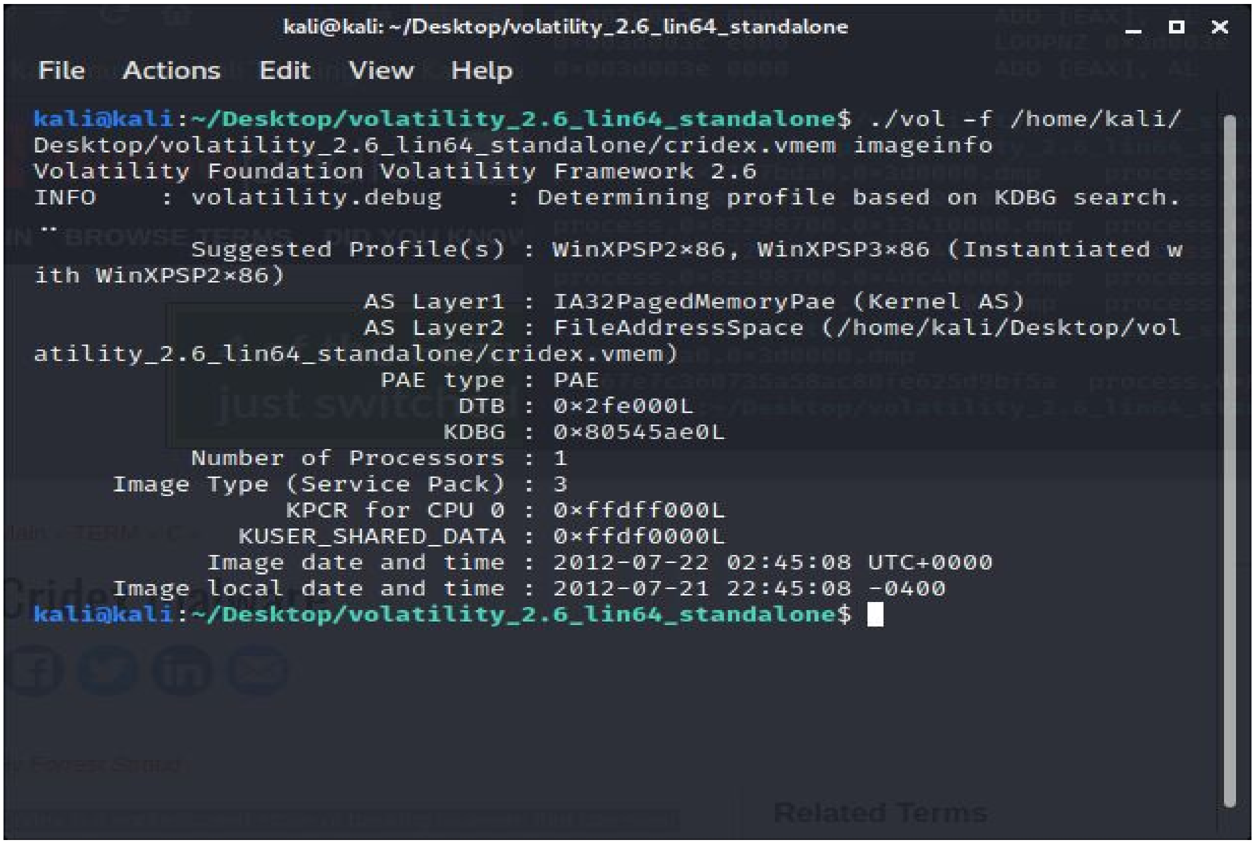
```
kali@kali:~/Desktop/volatility_2.6_lin64_standalone$ ./vol -h
Volatility Foundation Volatility Framework 2.6
Usage: Volatility - A memory forensics analysis platform.

Options:
  -h, --help            list all available options and their default values.
                        Default values may be set in the configuration file
                        (/etc/volatilityrc)
  --conf-file=/home/kali/.volatilityrc
                        User based configuration file
  -d, --debug           Debug volatility
  --plugins=PLUGINS     Additional plugin directories to use (colon separated)
  --info                Print information about all registered objects
  --cache-directory=/home/kali/.cache/volatility
                        Directory where cache files are stored
  --cache               Use caching
  --tz=TZ               Sets the (Olson) timezone for displaying timestamps
                        using pytz (if installed) or tzset
  -f FILENAME, --filename=FILENAME
                        Filename to use when opening an image
  --profile=WinXPSP2x86
                        Name of the profile to load (use --info to see a list
                        of supported profiles)
  -l LOCATION, --location=LOCATION
                        A URN location from which to load an address space
  -w, --write           Enable write support
  --dtb=DTB             DTB Address
  --shift=SHIFT         Mac KASLR shift address
  --output=text         Output in this format (support is module specific, see
                        the Module Output Options below)
  --output-file=OUTPUT_FILE
                        Write output in this file
  -v, --verbose         Verbose information
  -g KDBG, --kdbg=KDBG  Specify a KDBG virtual address (Note: for 64-bit
                        Windows 8 and above this is the address of
                        KdCopyDataBlock)
  --force               Force utilization of suspect profile
  --cookie=COOKIE       Specify the address of nt!ObHeaderCookie (valid for
                        Windows 10 only)
  -k KPCR, --kpcr=KPCR  Specify a specific KPCR address

        Supported Plugin Commands:

                amcache         Print AmCache information
                apihooks        Detect API hooks in process and kernel memory
                atoms           Print session and window station atom tables
                atomscan        Pool scanner for atom tables
                auditpol        Prints out the Audit Policies from HKLM\SECURITY\Policy\PolAdtEv
                bigpools        Dump the big page pools using BigPagePoolScanner
                bioskbd         Reads the keyboard buffer from Real Mode memory
                cachedump       Dumps cached domain hashes from memory
                callbacks       Print system-wide notification routines
                clipboard       Extract the contents of the windows clipboard
                cmdline         Display process command-line arguments
                cmdscan         Extract command history by scanning for _COMMAND_HISTORY
                connections     Print list of open connections [Windows XP and 2003 Only]
                connscan        Pool scanner for tcp connections
                consoles        Extract command history by scanning for _CONSOLE_INFORMATION
                crashinfo       Dump crash-dump information
                deskscan        Poolscaner for tagDESKTOP (desktops)
                devicetree      Show device tree
                dlldump         Dump DLLs from a process address space
                dlllist         Print list of loaded dlls for each process
                driverirp       Driver IRP hook detection
                drivermodule    Associate driver objects to kernel modules
                driverscan      Pool scanner for driver objects
                dumpcerts       Dump RSA private and public SSL keys
                dumpfiles       Extract memory mapped and cached files
                dumpregistry    Dumps registry files out to disk
                editbox         Displays information about Edit controls. (Listbox experimental.)
                envars          Display process environment variables
                eventhooks      Print details on windows event hooks
                evtlogs         Extract Windows Event Logs (XP/2003 only)
                filescan        Pool scanner for file objects
                gahti           Dump the USER handle type information
                gditimers       Print installed GDI timers and callbacks
```

Now that we have volatility working we need memory dumps to inspect. For this, the Volatility Foundation provides a plethora of memory samples taken from infected operating systems which you can download from their github wiki. For this tutorial and use case I chose to use the memory sample that is infected with Cridex. “Cridex is a sophisticated strain of banking malware that can steal banking credentials and other personal information on an infected system in order to gain access to the financial records of a user.” (Stroud, Cridex malware 2020)

Volatility Plugins

Volatility comes with a large assortment of plugins crucial to analyzing memory dumps and while using the argument -h, you can see all of those available and

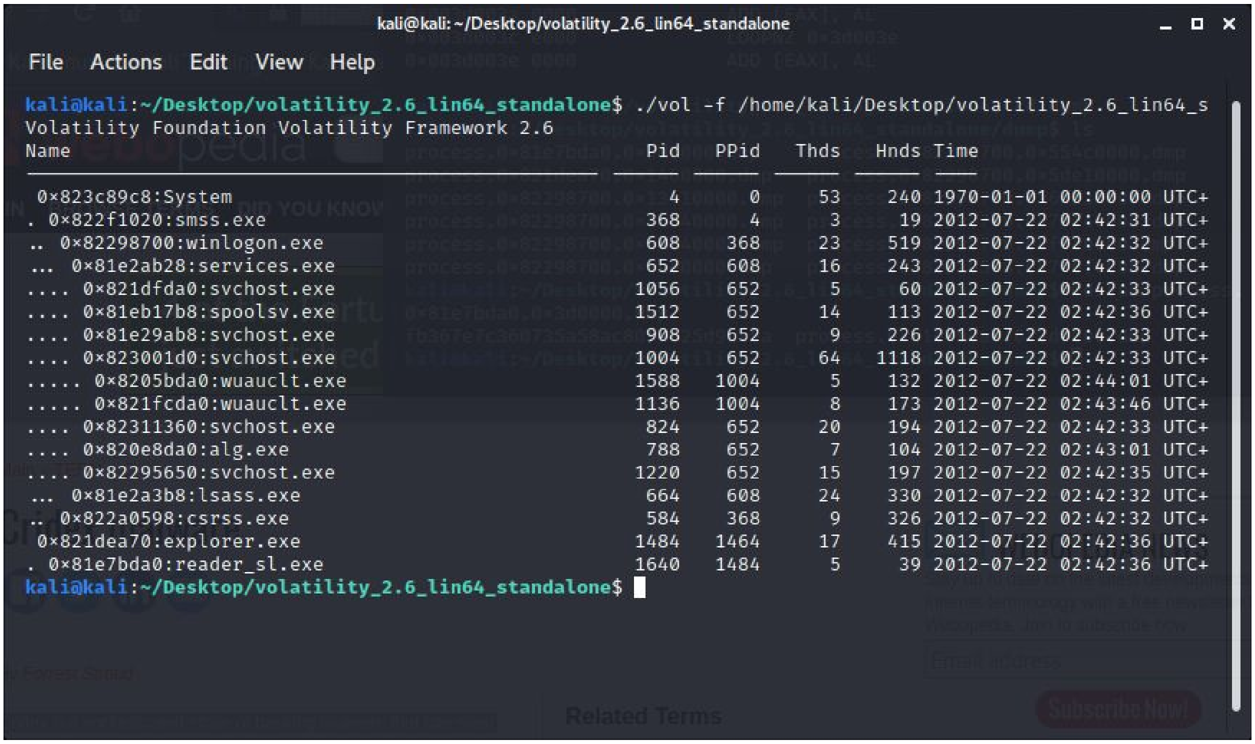
Imageinfo: Identify information for the image


kali@kali: ~/Desktop/volatility_2.6_lin64_standalone
File Actions Edit View Help

```
kali@kali:~/Desktop/volatility_2.6_lin64_standalone$ ./vol -f /home/kali/
Desktop/volatility_2.6_lin64_standalone/cridex.vmem imageinfo
Volatility Foundation Volatility Framework 2.6
INFO    : volatility.debug    : Determining profile based on KDBG search.
..
          Suggested Profile(s) : WinXPSP2x86, WinXPSP3x86 (Instantiated w
ith WinXPSP2x86)
                     AS Layer1 : IA32PagedMemoryPae (Kernel AS)
                     AS Layer2 : FileAddressSpace (/home/kali/Desktop/vol
atility_2.6_lin64_standalone/cridex.vmem)
                      PAE type : PAE
                           DTB : 0x2fe000L
                          KDBG : 0x80545ae0L
          Number of Processors : 1
     Image Type (Service Pack) : 3
                KPCR for CPU 0 : 0xffdff000L
             KUSER_SHARED_DATA : 0xffdf0000L
           Image date and time : 2012-07-22 02:45:08 UTC+0000
     Image local date and time : 2012-07-21 22:45:08 -0400
kali@kali:~/Desktop/volatility_2.6_lin64_standalone$ 
```

With image info we are returned with suggested profile for analyzing this memory and in this case the suggested profile is winXPSP2x86 and as well as this we can see this image was taken in 2012

Pstree: Shows process tree active on ram during the time of the capture



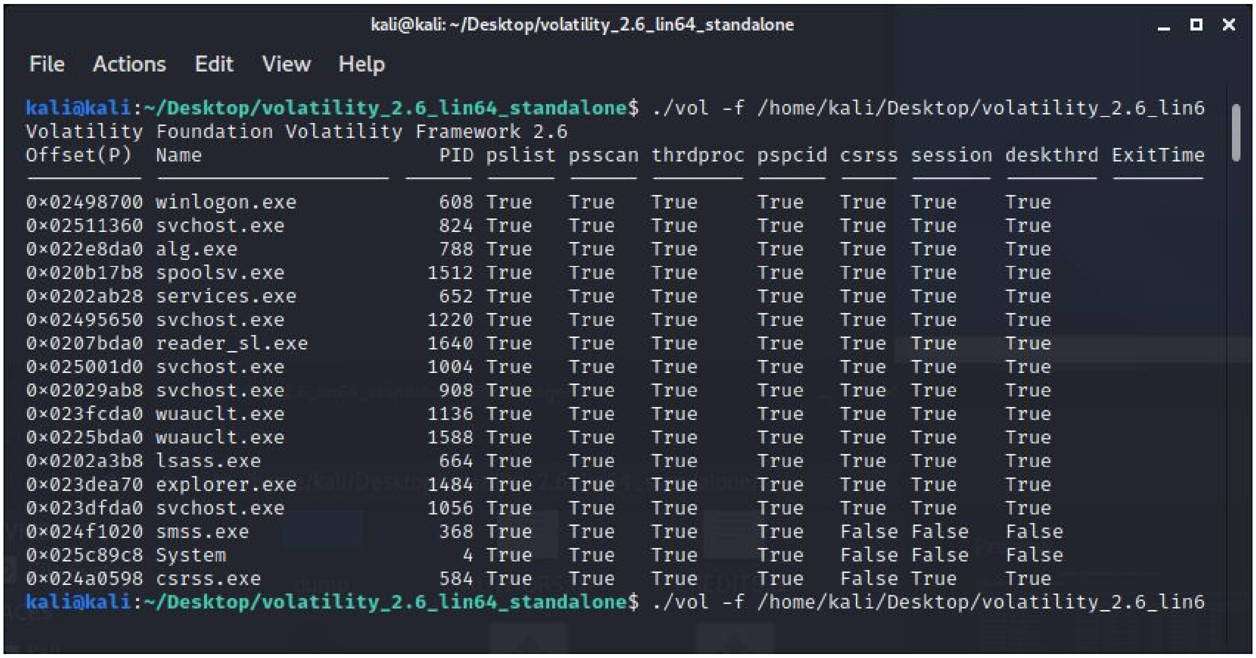
```
kali@kali:~/Desktop/volatility_2.6_lin64_standalone$ ./vol -f /home/kali/Desktop/volatility_2.6_lin64_s
Volatility Foundation Volatility Framework 2.6
Name                                                  Pid   PPid   Thds   Hnds Time
-------------------------------------------------- ------ ------ ------ ------ ----
 0x823c89c8:System                                      4      0     53    240 1970-01-01 00:00:00 UTC+
. 0x822f1020:smss.exe                                 368      4      3     19 2012-07-22 02:42:31 UTC+
.. 0x82298700:winlogon.exe                            608    368     23    519 2012-07-22 02:42:32 UTC+
... 0x81e2ab28:services.exe                           652    608     16    243 2012-07-22 02:42:32 UTC+
.... 0x821dfda0:svchost.exe                          1056    652      5     60 2012-07-22 02:42:33 UTC+
.... 0x81eb17b8:spoolsv.exe                          1512    652     14    113 2012-07-22 02:42:36 UTC+
.... 0x81e29ab8:svchost.exe                           908    652      9    226 2012-07-22 02:42:33 UTC+
.... 0x823001d0:svchost.exe                          1004    652     64   1118 2012-07-22 02:42:33 UTC+
..... 0x8205bda0:wuauclt.exe                         1588   1004      5    132 2012-07-22 02:44:01 UTC+
..... 0x821fcda0:wuauclt.exe                         1136   1004      8    173 2012-07-22 02:43:46 UTC+
.... 0x82311360:svchost.exe                           824    652     20    194 2012-07-22 02:42:33 UTC+
.... 0x820e8da0:alg.exe                               788    652      7    104 2012-07-22 02:43:01 UTC+
.... 0x82295650:svchost.exe                          1220    652     15    197 2012-07-22 02:42:35 UTC+
... 0x81e2a3b8:lsass.exe                              664    608     24    330 2012-07-22 02:42:32 UTC+
.. 0x822a0598:csrss.exe                               584    368      9    326 2012-07-22 02:42:32 UTC+
 0x821dea70:explorer.exe                             1484   1464     17    415 2012-07-22 02:42:36 UTC+
. 0x81e7bda0:reader_sl.exe                           1640   1484      5     39 2012-07-22 02:42:36 UTC+
kali@kali:~/Desktop/volatility_2.6_lin64_standalone$ 
```

A full example usage of pstree looks like:

./vol -f/home/kali/Desktop/volatility_2.6_lin64_standalone/tigger.vmem

--profile=WinXPSP2x86 pstree, where the profile pertains to the operating system and its file and memory structure and pstree is the plugin that allows us to see the captured active process tree. Even with this small process tree, we get some very important information, that being the PID (process identifier). This identifier can help us identify what may be sending data over the internet or in general what interactions with the computer have been captured.

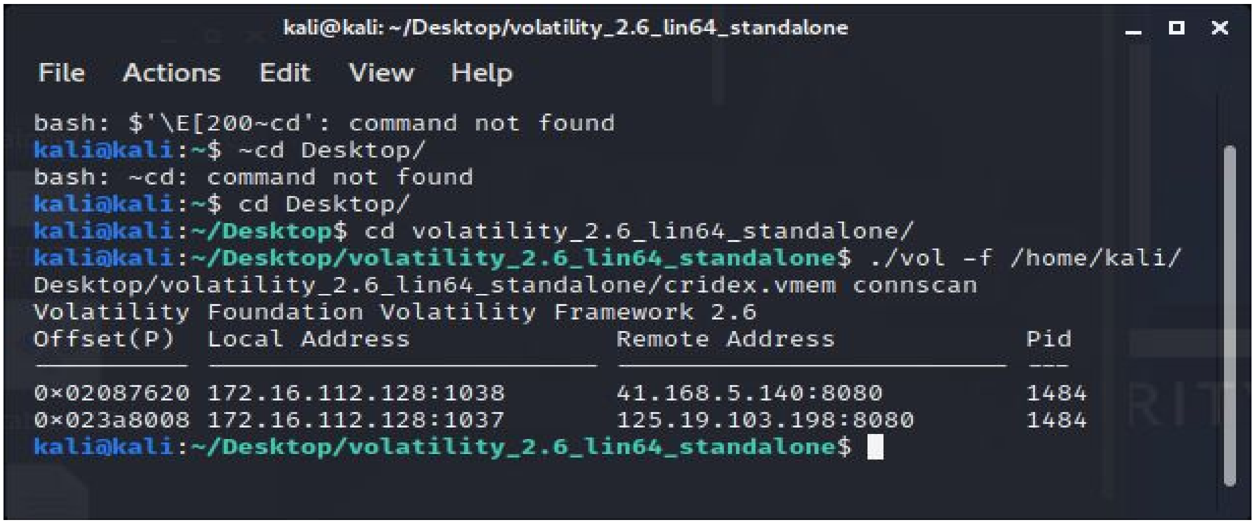
Psxview: Find Hidden processes with their process listings



```
kali@kali:~/Desktop/volatility_2.6_lin64_standalone$ ./vol -f /home/kali/Desktop/volatility_2.6_lin6
Volatility Foundation Volatility Framework 2.6
Offset(P)  Name                    PID pslist psscan thrdproc pspcid csrss session deskthrd ExitTime
---------- -------------------- ------ ------ ------ -------- ------ ----- ------- -------- --------
0x02498700 winlogon.exe            608 True   True   True     True   True  True    True
0x02511360 svchost.exe             824 True   True   True     True   True  True    True
0x022e8da0 alg.exe                 788 True   True   True     True   True  True    True
0x020b17b8 spoolsv.exe            1512 True   True   True     True   True  True    True
0x0202ab28 services.exe            652 True   True   True     True   True  True    True
0x02495650 svchost.exe            1220 True   True   True     True   True  True    True
0x0207bda0 reader_sl.exe          1640 True   True   True     True   True  True    True
0x025001d0 svchost.exe            1004 True   True   True     True   True  True    True
0x02029ab8 svchost.exe             908 True   True   True     True   True  True    True
0x023fcda0 wuauclt.exe            1136 True   True   True     True   True  True    True
0x0225bda0 wuauclt.exe            1588 True   True   True     True   True  True    True
0x0202a3b8 lsass.exe               664 True   True   True     True   True  True    True
0x023dea70 explorer.exe           1484 True   True   True     True   True  True    True
0x023dfda0 svchost.exe            1056 True   True   True     True   True  True    True
0x024f1020 smss.exe                368 True   True   True     True   False False   False
0x025c89c8 System                    4 True   True   True     True   False False   False
0x024a0598 csrss.exe               584 True   True   True     True   False True    True
kali@kali:~/Desktop/volatility_2.6_lin64_standalone$ ./vol -f /home/kali/Desktop/volatility_2.6_lin6
```

Psxview can give great insight into what is running on our machine. Greater than pstree, it allows us to see what processes may be trying to hide so long as they come up false in both pslist and psscan sections. As we can see nothing is coming up false here.

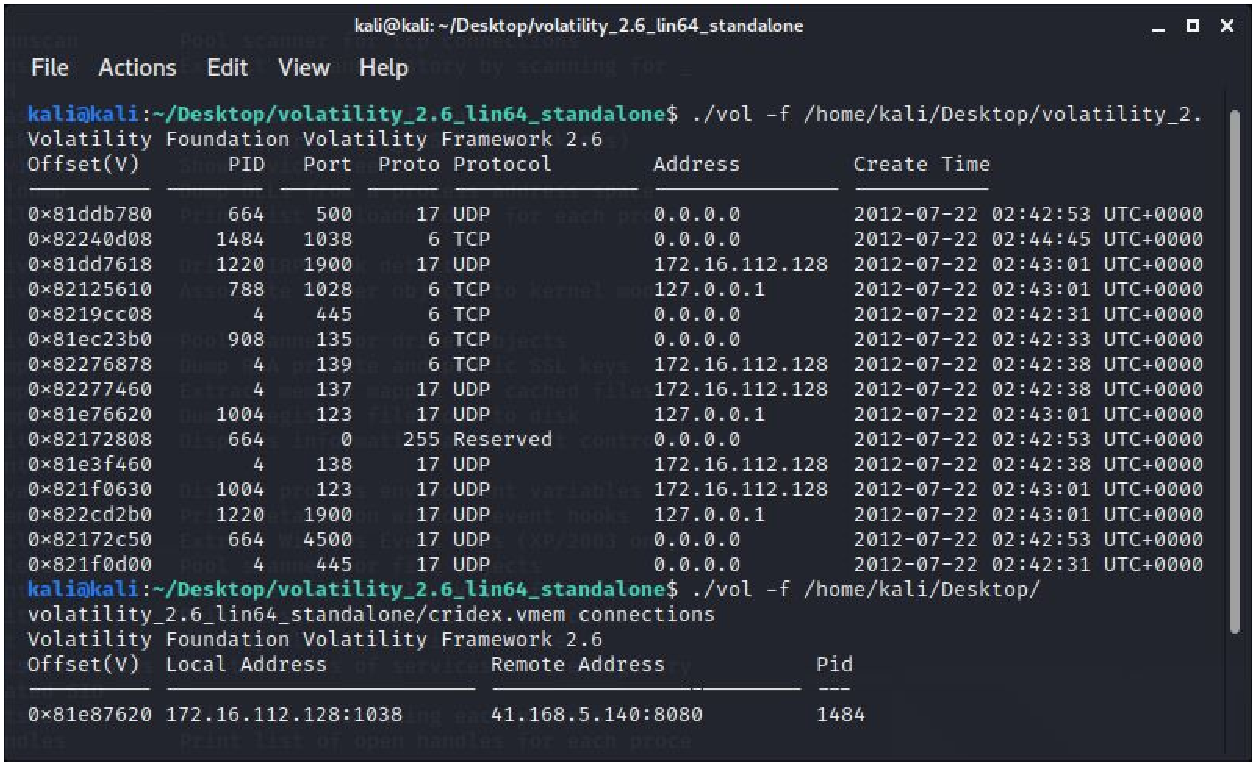
Connscan: Pool scanner for tcp connections



```
bash: $'\E[200~cd': command not found
kali@kali:~$ ~cd Desktop/
bash: ~cd: command not found
kali@kali:~$ cd Desktop/
kali@kali:~/Desktop$ cd volatility_2.6_lin64_standalone/
kali@kali:~/Desktop/volatility_2.6_lin64_standalone$ ./vol -f /home/kali/
Desktop/volatility_2.6_lin64_standalone/cridex.vmem connscan
Volatility Foundation Volatility Framework 2.6
Offset(P)  Local Address             Remote Address            Pid
---------- ------------------------- ------------------------- ---
0x02087620 172.16.112.128:1038       41.168.5.140:8080         1484
0x023a8008 172.16.112.128:1037       125.19.103.198:8080       1484
kali@kali:~/Desktop/volatility_2.6_lin64_standalone$
```

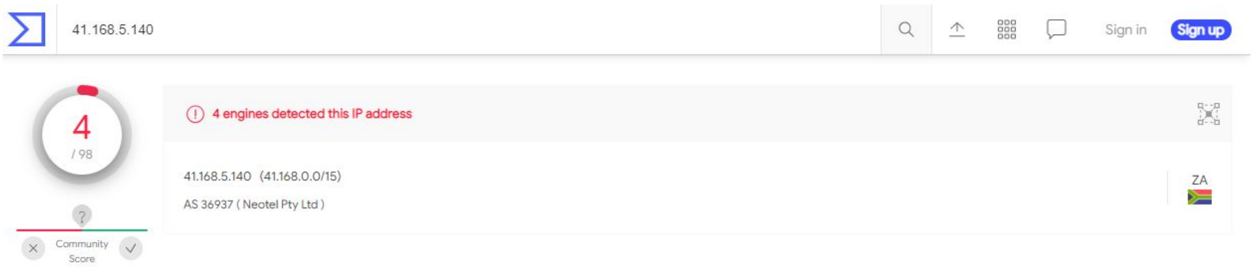
Connscan gives us great insight on what processes are transmitting over tcp connections as we can see here the process associated with these transmissions belongs to process ID 1484. As we can see above this image, 1484 is the process of explorer.exe.

Connections and Sockets:



```
kali@kali:~/Desktop/volatility_2.6_lin64_standalone$ ./vol -f /home/kali/Desktop/volatility_2.
Volatility Foundation Volatility Framework 2.6
Offset(V)         PID   Port  Proto Protocol        Address         Create Time
---------- -------- ------ ------ --------------- --------------- -----------
0×81ddb780      664    500     17 UDP             0.0.0.0         2012-07-22 02:42:53 UTC+0000
0×82240d08     1484   1038      6 TCP             0.0.0.0         2012-07-22 02:44:45 UTC+0000
0×81dd7618     1220   1900     17 UDP             172.16.112.128  2012-07-22 02:43:01 UTC+0000
0×82125610      788   1028      6 TCP             127.0.0.1       2012-07-22 02:43:01 UTC+0000
0×8219cc08        4    445      6 TCP             0.0.0.0         2012-07-22 02:42:31 UTC+0000
0×81ec23b0      908    135      6 TCP             0.0.0.0         2012-07-22 02:42:33 UTC+0000
0×82276878        4    139      6 TCP             172.16.112.128  2012-07-22 02:42:38 UTC+0000
0×82277460        4    137     17 UDP             172.16.112.128  2012-07-22 02:42:38 UTC+0000
0×81e76620     1004    123     17 UDP             127.0.0.1       2012-07-22 02:43:01 UTC+0000
0×82172808      664      0    255 Reserved        0.0.0.0         2012-07-22 02:42:53 UTC+0000
0×81e3f460        4    138     17 UDP             172.16.112.128  2012-07-22 02:42:38 UTC+0000
0×821f0630     1004    123     17 UDP             172.16.112.128  2012-07-22 02:43:01 UTC+0000
0×822cd2b0     1220   1900     17 UDP             127.0.0.1       2012-07-22 02:43:01 UTC+0000
0×82172c50      664   4500     17 UDP             0.0.0.0         2012-07-22 02:42:53 UTC+0000
0×821f0d00        4    445     17 UDP             0.0.0.0         2012-07-22 02:42:31 UTC+0000
kali@kali:~/Desktop/volatility_2.6_lin64_standalone$ ./vol -f /home/kali/Desktop/
volatility_2.6_lin64_standalone/cridex.vmem connections
Volatility Foundation Volatility Framework 2.6
Offset(V)  Local Address             Remote Address            Pid
---------- ------------------------- ------------------------- ---
0×81e87620 172.16.112.128:1038       41.168.5.140:8080         1484
```

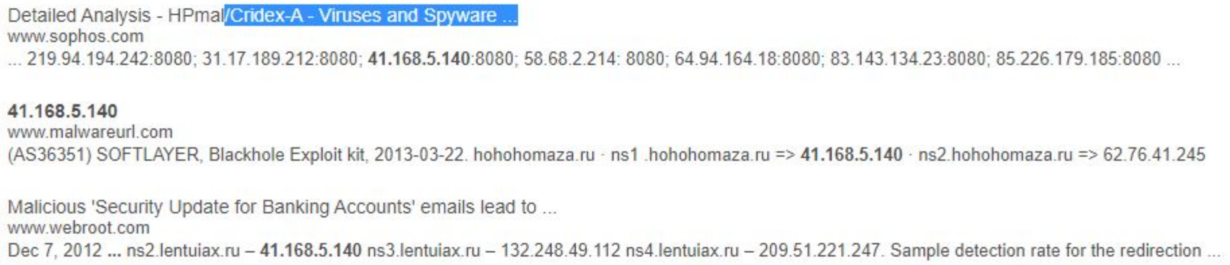
As well as conscan, we have connections and sockets. Both prove to be incredibly important as well as connections show us all of the open connections during the time of capture and sockets show sockets in use. Within the reported section from connections I am immediately intrigued by the open connection on explorer. This IP under remote address can be from anywhere but thankfully we only have one open connection as looking at many ip's could be tedious. To get a closer look at what this IP is we can use a site called VirusTotal. "VirusTotal inspects items with over 70 antivirus scanners and URL/domain blacklisting services, in addition to a myriad of tools to extract signals from the studied content” (VirusTotal, How it works 2020) With this at our aid, gaining insight on that ip address should be easier.



4 engines detected this IP address

41.168.5.140 (41.168.0.0/15)
AS 36937 (Neotel Pty Ltd)

With that search we now see that it comes up as related to malware in some capacity. In fact, VirusTotal allows us to see more detail including where this ip address has popped up in articles or reports and we can see that it directly links to Cridex, a blackhole exploit kit, and banking account emails leading to the black hole exploit kit.



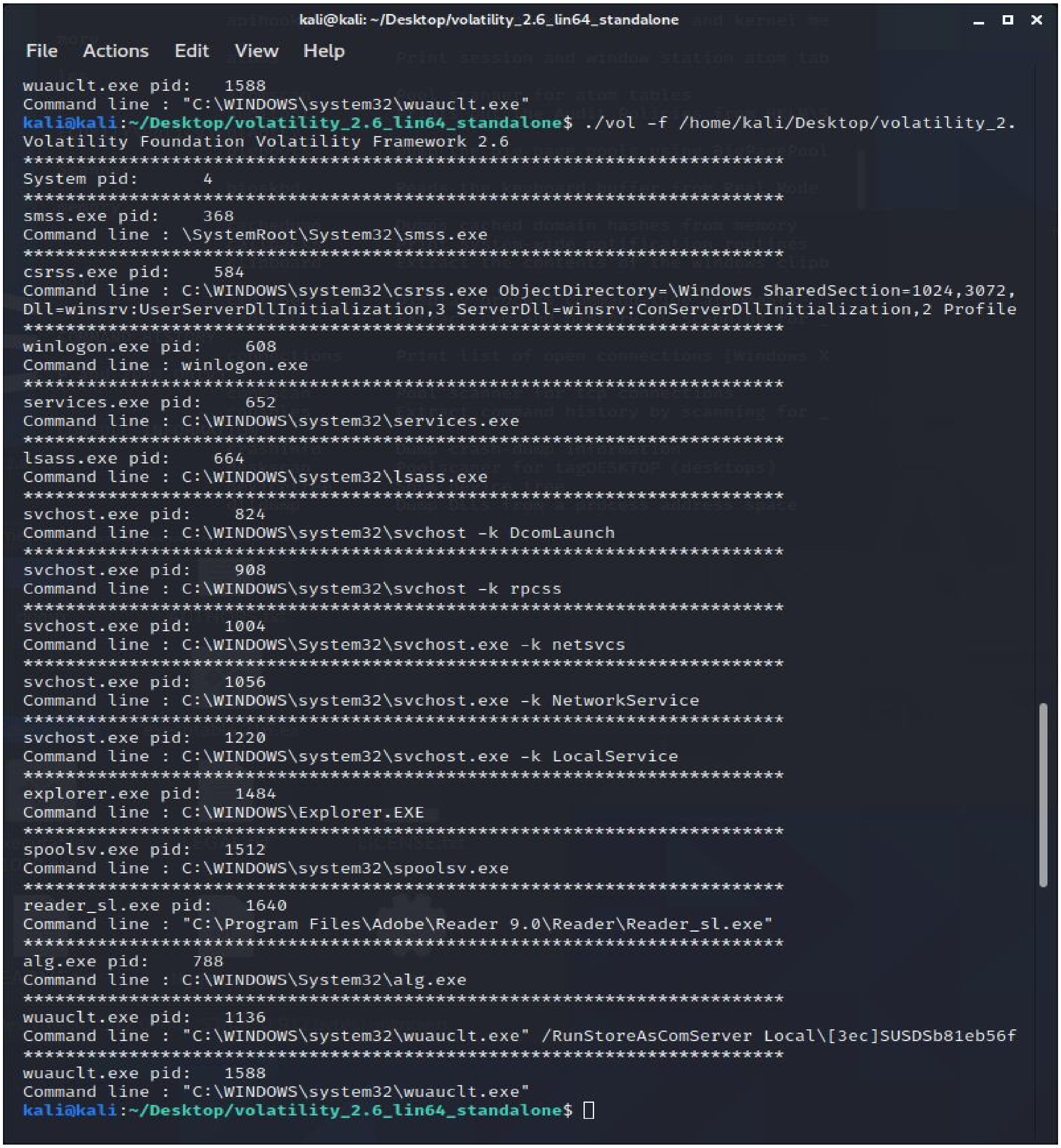
Detailed Analysis - HPmal/Cridex-A - Viruses and Spyware ...
www.sophos.com
... 219.94.194.242:8080; 31.17.189.212:8080; **41.168.5.140**:8080; 58.68.2.214: 8080; 64.94.164.18:8080; 83.143.134.23:8080; 85.226.179.185:8080 ...

41.168.5.140
www.malwareurl.com
(AS36351) SOFTLAYER, Blackhole Exploit kit, 2013-03-22. hohohomaza.ru · ns1 .hohohomaza.ru => **41.168.5.140** · ns2.hohohomaza.ru => 62.76.41.245

Malicious 'Security Update for Banking Accounts' emails lead to ...
www.webroot.com
Dec 7, 2012 **...** ns2.lentuiax.ru – **41.168.5.140** ns3.lentuiax.ru – 132.248.49.112 ns4.lentuiax.ru – 209.51.221.247. Sample detection rate for the redirection ...

(Virus Total, 2020).

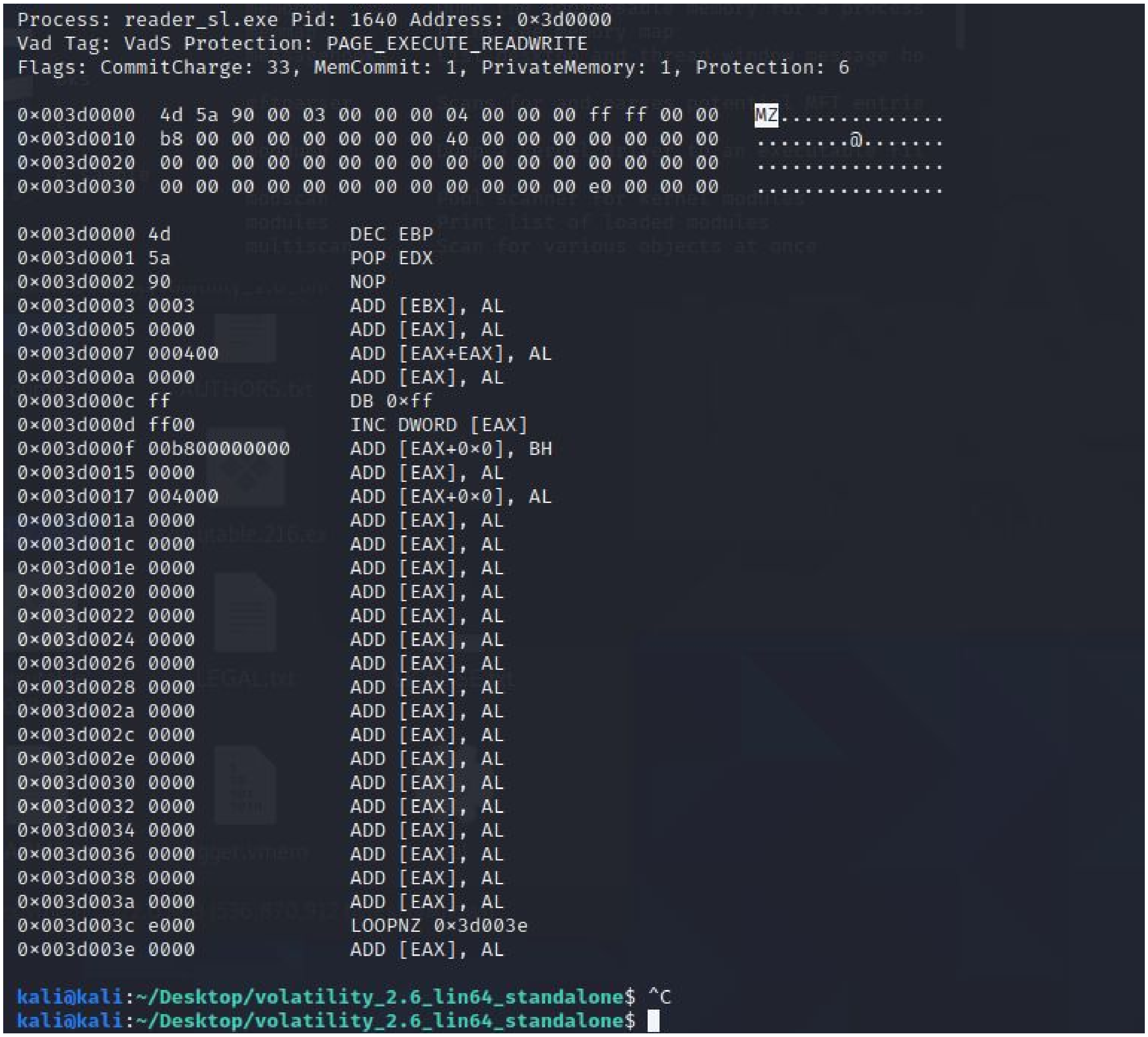
Cmdline: Display process command-line arguments



```
wuauclt.exe pid:   1588
Command line : "C:\WINDOWS\system32\wuauclt.exe"
kali@kali:~/Desktop/volatility_2.6_lin64_standalone$ ./vol -f /home/kali/Desktop/volatility_2.
Volatility Foundation Volatility Framework 2.6
************************************************************************
System pid:      4
************************************************************************
smss.exe pid:    368
Command line : \SystemRoot\System32\smss.exe
************************************************************************
csrss.exe pid:    584
Command line : C:\WINDOWS\system32\csrss.exe ObjectDirectory=\Windows SharedSection=1024,3072,
Dll=winsrv:UserServerDllInitialization,3 ServerDll=winsrv:ConServerDllInitialization,2 Profile
************************************************************************
winlogon.exe pid:    608
Command line : winlogon.exe
************************************************************************
services.exe pid:    652
Command line : C:\WINDOWS\system32\services.exe
************************************************************************
lsass.exe pid:    664
Command line : C:\WINDOWS\system32\lsass.exe
************************************************************************
svchost.exe pid:    824
Command line : C:\WINDOWS\system32\svchost -k DcomLaunch
************************************************************************
svchost.exe pid:    908
Command line : C:\WINDOWS\system32\svchost -k rpcss
************************************************************************
svchost.exe pid:   1004
Command line : C:\WINDOWS\System32\svchost.exe -k netsvcs
************************************************************************
svchost.exe pid:   1056
Command line : C:\WINDOWS\system32\svchost.exe -k NetworkService
************************************************************************
svchost.exe pid:   1220
Command line : C:\WINDOWS\system32\svchost.exe -k LocalService
************************************************************************
explorer.exe pid:   1484
Command line : C:\WINDOWS\Explorer.EXE
************************************************************************
spoolsv.exe pid:   1512
Command line : C:\WINDOWS\system32\spoolsv.exe
************************************************************************
reader_sl.exe pid:   1640
Command line : "C:\Program Files\Adobe\Reader 9.0\Reader\Reader_sl.exe"
************************************************************************
alg.exe pid:    788
Command line : C:\WINDOWS\System32\alg.exe
************************************************************************
wuauclt.exe pid:   1136
Command line : "C:\WINDOWS\system32\wuauclt.exe" /RunStoreAsComServer Local\[3ec]SUSDSb81eb56f
************************************************************************
wuauclt.exe pid:   1588
Command line : "C:\WINDOWS\system32\wuauclt.exe"
kali@kali:~/Desktop/volatility_2.6_lin64_standalone$
```

Now that we have insight on the open connection, looking at the captured command line arguments allow us to see when things started running and in which order. We know that explorer is running on an open connection that is associated with malware, we can see that Reader_sl comes after and becomes more and more suspicious.

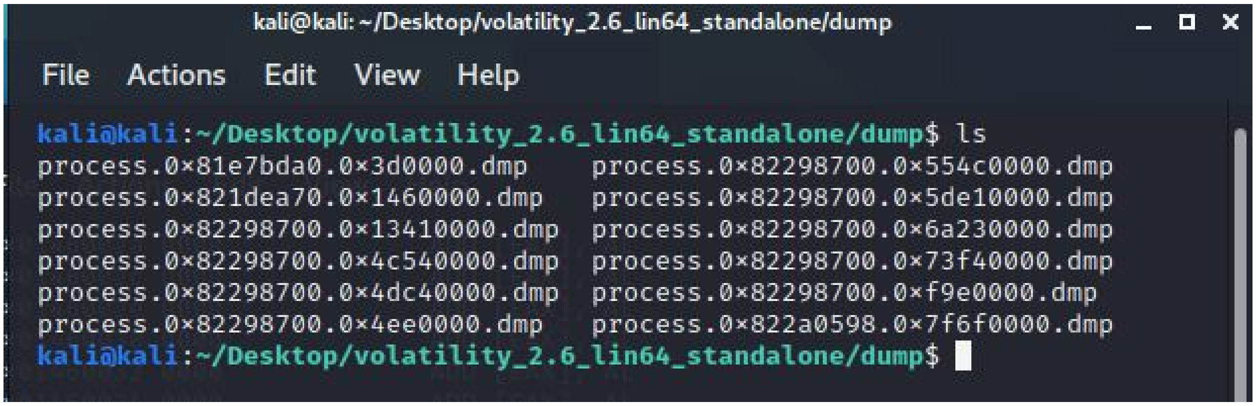
Malfind: Find hidden injected code



```
Process: reader_sl.exe Pid: 1640 Address: 0x3d0000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 33, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x003d0000  4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00   MZ..............
0x003d0010  b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00   ........@.......
0x003d0020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x003d0030  00 00 00 00 00 00 00 00 00 00 00 00 e0 00 00 00   ................

0x003d0000 4d               DEC EBP
0x003d0001 5a               POP EDX
0x003d0002 90               NOP
0x003d0003 0003             ADD [EBX], AL
0x003d0005 0000             ADD [EAX], AL
0x003d0007 000400           ADD [EAX+EAX], AL
0x003d000a 0000             ADD [EAX], AL
0x003d000c ff               DB 0xff
0x003d000d ff00             INC DWORD [EAX]
0x003d000f 00b800000000     ADD [EAX+0x0], BH
0x003d0015 0000             ADD [EAX], AL
0x003d0017 004000           ADD [EAX+0x0], AL
0x003d001a 0000             ADD [EAX], AL
0x003d001c 0000             ADD [EAX], AL
0x003d001e 0000             ADD [EAX], AL
0x003d0020 0000             ADD [EAX], AL
0x003d0022 0000             ADD [EAX], AL
0x003d0024 0000             ADD [EAX], AL
0x003d0026 0000             ADD [EAX], AL
0x003d0028 0000             ADD [EAX], AL
0x003d002a 0000             ADD [EAX], AL
0x003d002c 0000             ADD [EAX], AL
0x003d002e 0000             ADD [EAX], AL
0x003d0030 0000             ADD [EAX], AL
0x003d0032 0000             ADD [EAX], AL
0x003d0034 0000             ADD [EAX], AL
0x003d0036 0000             ADD [EAX], AL
0x003d0038 0000             ADD [EAX], AL
0x003d003a 0000             ADD [EAX], AL
0x003d003c e000             LOOPNZ 0x3d003e
0x003d003e 0000             ADD [EAX], AL

kali@kali:~/Desktop/volatility_2.6_lin64_standalone$ ^C
kali@kali:~/Desktop/volatility_2.6_lin64_standalone$ 
```

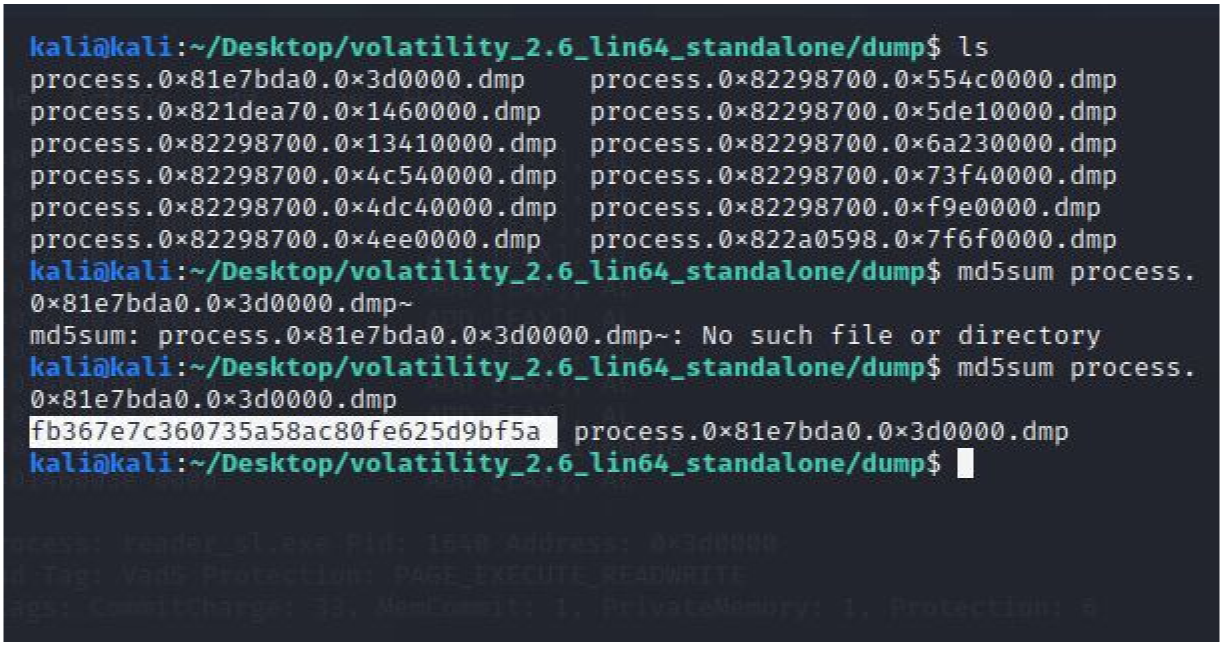
With malfind at our disposal we are able to see processes with injected memory. With reader coming up as injected we should dump the process for further inspection with ./vol -f /home/kali/Desktop/volatility_2.6_lin64_standalone/cridex.vmem malfind --dump-dir=dump/

Where dump/ is the location of where I want to dump the output.


kali@kali: ~/Desktop/volatility_2.6_lin64_standalone/dump
File Actions Edit View Help

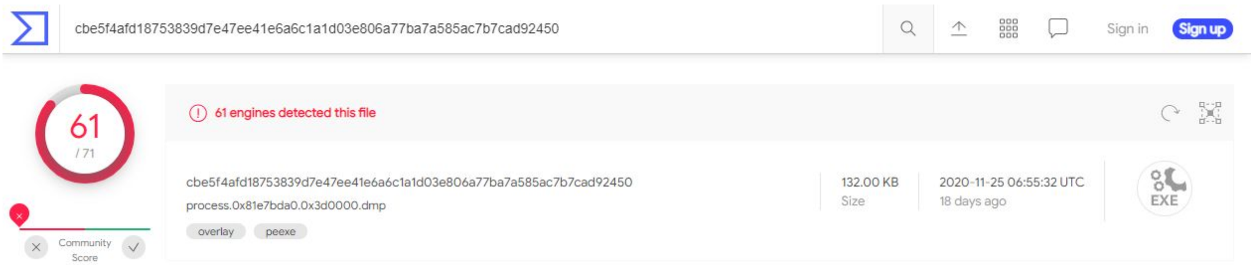
```
kali@kali:~/Desktop/volatility_2.6_lin64_standalone/dump$ ls
process.0x81e7bda0.0x3d0000.dmp    process.0x82298700.0x554c0000.dmp
process.0x821dea70.0x1460000.dmp   process.0x82298700.0x5de10000.dmp
process.0x82298700.0x13410000.dmp  process.0x82298700.0x6a230000.dmp
process.0x82298700.0x4c540000.dmp  process.0x82298700.0x73f40000.dmp
process.0x82298700.0x4dc40000.dmp  process.0x82298700.0xf9e0000.dmp
process.0x82298700.0x4ee0000.dmp   process.0x822a0598.0x7f6f0000.dmp
kali@kali:~/Desktop/volatility_2.6_lin64_standalone/dump$
```

Now that we have all of the processes in the format of a dmp file, what we can then do next is get the hash of one of these files using md5sum. Md5 is a widely used hashing algorithm and its output is one to one; meaning that if I get the hash of anyone of this file it will give me the same hash as anyone to do the same hash to these dmp files.



```
kali@kali:~/Desktop/volatility_2.6_lin64_standalone/dump$ ls
process.0x81e7bda0.0x3d0000.dmp    process.0x82298700.0x554c0000.dmp
process.0x821dea70.0x1460000.dmp   process.0x82298700.0x5de10000.dmp
process.0x82298700.0x13410000.dmp  process.0x82298700.0x6a230000.dmp
process.0x82298700.0x4c540000.dmp  process.0x82298700.0x73f40000.dmp
process.0x82298700.0x4dc40000.dmp  process.0x82298700.0xf9e0000.dmp
process.0x82298700.0x4ee0000.dmp   process.0x822a0598.0x7f6f0000.dmp
kali@kali:~/Desktop/volatility_2.6_lin64_standalone/dump$ md5sum process.
0x81e7bda0.0x3d0000.dmp~
md5sum: process.0x81e7bda0.0x3d0000.dmp~: No such file or directory
kali@kali:~/Desktop/volatility_2.6_lin64_standalone/dump$ md5sum process.
0x81e7bda0.0x3d0000.dmp
fb367e7c360735a58ac80fe625d9bf5a  process.0x81e7bda0.0x3d0000.dmp
kali@kali:~/Desktop/volatility_2.6_lin64_standalone/dump$
```

Here we have the hash generated of the process and using this we can go back to VirusTotal and see if it is recognized.



cbe5f4afd18753839d7e47ee41e6a6c1a1d03e806a77ba7a585ac7b7cad92450

61 engines detected this file

cbe5f4afd18753839d7e47ee41e6a6c1a1d03e806a77ba7a585ac7b7cad92450
process.0x81e7bda0.0x3d0000.dmp
overlay peexe

132.00 KB
Size

2020-11-25 06:55:32 UTC
18 days ago

As it seems we have found the culprit within the processes of Reader_Sl. Recognized by 61 engines we have been able to isolate the location of Cridex on this system.

Conclusion

The analysis of volatile memory for signs of infection is integral to our understanding of computer security as there are forms of attacks that basic virus defenders are blind to.Volatility provides tools crucial to the study of memory forensics and the active use of memory forensics to defend computing systems in private or public settings. It has become understood that many, more sophisticated trojans hide themselves in many ways. While Cridex did not hide itself in process view, it hid itself within an existing adobe software. Using volatility to analyze captured memory has allowed me to gain knowledge on how to recognize rogue processes and secure my system from them.

References

- Memory forensics. (2020, April 23). Retrieved october 10, 2020, from https://en.wikipedia.org/wiki/Memory_forensics
- Volatility (memory forensics). (2018, December 03). Retrieved october 10, 2020, from https://en.wikipedia.org/wiki/Volatility_(memory_forensics)

- Duc, H. (2016, June 29). Finding Advanced Malware Using Volatility. Retrieved October 30, 2020, from https://eforensicsmag.com/finding-advanced-malware-using-volatility/
- Wikipedia contributors. (2020, November 30). *Fileless malware*. Wikipedia. https://en.wikipedia.org/wiki/Fileless_malware
- Stroud, F. (n.d.). Cridex malware. Retrieved December 13, 2020, from https://www.webopedia.com/TERM/C/cridex-malware.html
- How it works. (n.d.). Retrieved December 13, 2020, from https://support.virustotal.com/hc/en-us/articles/115002126889-How-it-works
- Virus Total IP (n.d.). Retrieved December 13, 2020, from https://www.virustotal.com/gui/ip-address/41.168.5.140/details