Week 3 Lab Comp6445 By Claire Fei (z5161105)

PART 1

In this Forensic Analysis, we will be primarily using the tool volatility which is an advanced memory forensics framework to assist in the analysis of the given memory dump “cridex.vmem”.

Step 1: Generate an md5sum of the memory dump to ensure the integrity of the file.

```
root@kali:~/Documents/cridex# md5sum cridex.vmem
7494f3b77db1525c1974f5380744ae46  cridex.vmem
```

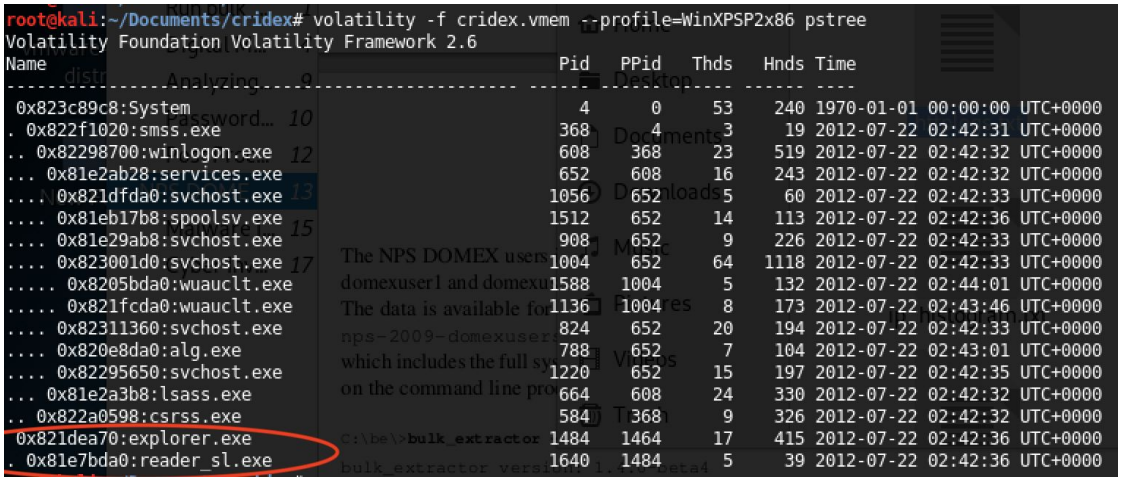
Step 2: We will now get some high-level information on the “cridex.vmem” memory dump by using the “imageinfo” command. This will allow us to determine useful information including the operation system/service pack as seen in the “Suggested Profile(s)” subheading. This will be useful for disassembling the memory dump as different operating systems and service packs have different structures and kernels which will affect the way the rest of our tools determine information whilst extracting details from the memory dump.

```
root@kali:~/Documents/cridex# volatility -f cridex.vmem imageinfo
Volatility Foundation Volatility Framework 2.6
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 (/root/Documents/cridex/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
```

Step 3: We will now use the “pslist” command to get a list of processes that are actively running. The pslist achieves this by looking at the process linked list structure and iterating over each one to see what processes the operating system is aware of. A lot of these such as System and svchost.exe which are default processes spawned by the Windows10 operating system are quite normal in the usual case, however, the two interesting processes are reader.sl.exe and alg.exe (circled below) which are usually user-spawned processes. As such, we will be paying closer attention to these.

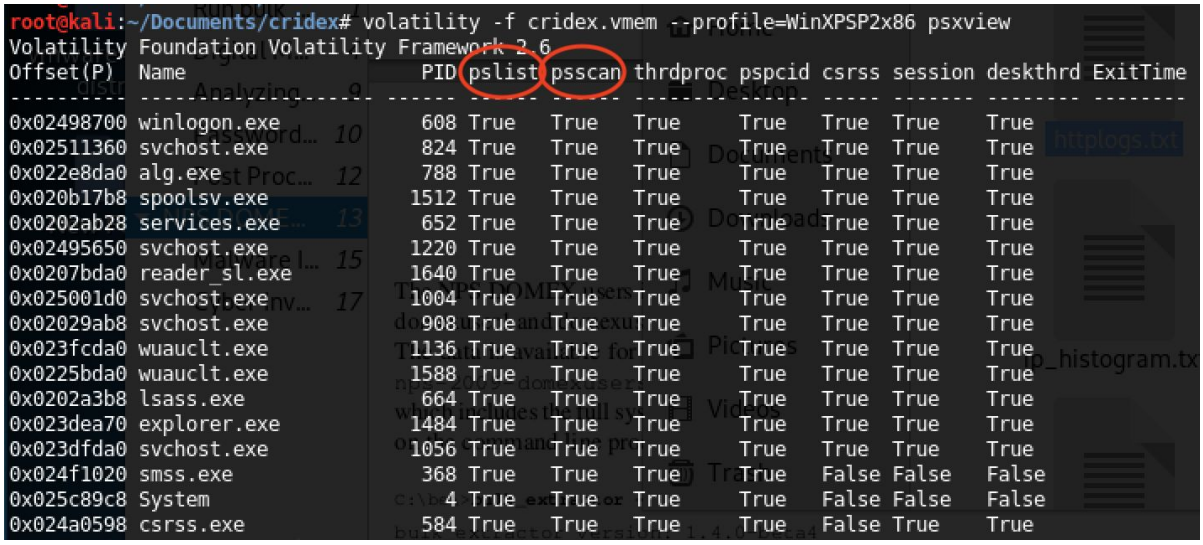
```
root@kali:~/Documents/cridex# volatility -f cridex.vmem --profile=WinXPSP2x86 pslist
Volatility Foundation Volatility Framework 2.6
Offset(V)  Name                    PID   PPID   Thds     Hnds   Sess  Wow64 Start                          Exit
---------- -------------------- ------ ------ ------ -------- ------ ------ ------------------------------ ------
0x823c89c8 System                    4      0     53      240 ------      0
0x822f1020 smss.exe                368      4      3       19 ------      0 2012-07-22 02:42:31 UTC+0000
0x822a0598 csrss.exe               584    368      9      326      0      0 2012-07-22 02:42:32 UTC+0000
0x82298700 winlogon.exe            608    368     23      519      0      0 2012-07-22 02:42:32 UTC+0000
0x81e2ab28 services.exe            652    608     16      243      0      0 2012-07-22 02:42:32 UTC+0000
0x81e2a3b8 lsass.exe               664    608     24      330      0      0 2012-07-22 02:42:32 UTC+0000
0x82311360 svchost.exe             824    652     20      194      0      0 2012-07-22 02:42:33 UTC+0000
0x81e29ab8 svchost.exe             908    652      9      226      0      0 2012-07-22 02:42:33 UTC+0000
0x823001d0 svchost.exe            1004    652     64     1118      0      0 2012-07-22 02:42:33 UTC+0000
0x821dfda0 svchost.exe            1056    652      5       60      0      0 2012-07-22 02:42:33 UTC+0000
0x82295650 svchost.exe            1220    652     15      197      0      0 2012-07-22 02:42:35 UTC+0000
0x821dea70 explorer.exe           1484   1464     17      415      0      0 2012-07-22 02:42:36 UTC+0000
0x81eb17b8 spoolsv.exe            1512    652     14      113      0      0 2012-07-22 02:42:36 UTC+0000
0x81e7bda0 reader_sl.exe          1640   1484      5       39      0      0 2012-07-22 02:42:36 UTC+0000
0x820e8da0 alg.exe                 788    652      7      104      0      0 2012-07-22 02:43:01 UTC+0000
0x821fcda0 wuauclt.exe            1136   1004      8      173      0      0 2012-07-22 02:43:46 UTC+0000
0x8205bda0 wuauclt.exe            1588   1004      5      132      0      0 2012-07-22 02:44:01 UTC+0000
```

Step 4: Using the “pstree” command places all the processes into a nice hierarchy, allowing us to see what processes are spawned by other processes in a nice visual format. The indentation is indicative of a child process, spawning from the unindented process. One important thing to note is that svchost.exe should be spawned by the System process, since it is in this analysis, we can not make any inference from this new information. Looking at the reader_sl.exe which is usually a process spawned by adobe reader, it has the explore.exe as a parent (file explorer). This is not suspicious either, so we will continue our search.



```
root@kali:~/Documents/cridex# volatility -f cridex.vmem --profile=WinXPSP2x86 pstree
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+0000
. 0x822f1020:smss.exe                                 368      4      3     19 2012-07-22 02:42:31 UTC+0000
.. 0x82298700:winlogon.exe                            608    368     23    519 2012-07-22 02:42:32 UTC+0000
... 0x81e2ab28:services.exe                           652    608     16    243 2012-07-22 02:42:32 UTC+0000
.... 0x821dfda0:svchost.exe                          1056    652      5     60 2012-07-22 02:42:33 UTC+0000
.... 0x81eb17b8:spoolsv.exe                          1512    652     14    113 2012-07-22 02:42:36 UTC+0000
.... 0x81e29ab8:svchost.exe                           908    652      9    226 2012-07-22 02:42:33 UTC+0000
.... 0x823001d0:svchost.exe                          1004    652     64   1118 2012-07-22 02:42:33 UTC+0000
..... 0x8205bda0:wuauclt.exe                         1588   1004      5    132 2012-07-22 02:44:01 UTC+0000
..... 0x821fcda0:wuauclt.exe                         1136   1004      8    173 2012-07-22 02:43:46 UTC+0000
.... 0x82311360:svchost.exe                           824    652     20    194 2012-07-22 02:42:33 UTC+0000
.... 0x820e8da0:alg.exe                               788    652      7    104 2012-07-22 02:43:01 UTC+0000
.... 0x82295650:svchost.exe                          1220    652     15    197 2012-07-22 02:42:35 UTC+0000
... 0x81e2a3b8:lsass.exe                              664    608     24    330 2012-07-22 02:42:32 UTC+0000
.. 0x822a0598:csrss.exe                               584    368      9    326 2012-07-22 02:42:32 UTC+0000
 0x821dea70:explorer.exe                             1484   1464     17    415 2012-07-22 02:42:36 UTC+0000
. 0x81e7bda0:reader_sl.exe                           1640   1484      5     39 2012-07-22 02:42:36 UTC+0000
```

Step 5: Another command we can use to look for suspicious activity is the “psxview” command. This command simply uses different process identification tools to see if the processes are detected. The main two tools we are interested in are pslist nad psscan. The important differentiation is that pslist actively identifies process from the kernel’s process linked list where as psscan scans the entirety of memory and identifies any process-looking structures. Psscan can often find closed/dead processes as a result of this methodology for process identification. The key thing we are looking for here is if there are any inconsistencies e.g “true” marking a process as found in pslist but “false” in psscan or vice versa.



```
root@kali:~/Documents/cridex# volatility -f cridex.vmem --profile=WinXPSP2x86 psxview
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
```

Step 6: Another thing we want to look at is “connscan” which looks for connections. These connections in themselves are not suspicious, but if we look at their process id and match it to the process, we can see that explorer.exe has two connections attached to it. This is suspicious as file explorer usually does not require a connection as it is mostly used as a local file opening tool.

```
root@kali:~/Documents/cridex# volatility -f cridex.vmem --profile=WinXPSP2x86 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
```

```
root@kali:~/Documents/cridex# volatility -f cridex.vmem --profile=WinXPSP3x86 psxview | egrep "explorer|Pid"
Volatility Foundation Volatility Framework 2.6
0x023dea70 explorer.exe          1484 True   True   True     True   True  True    True
```

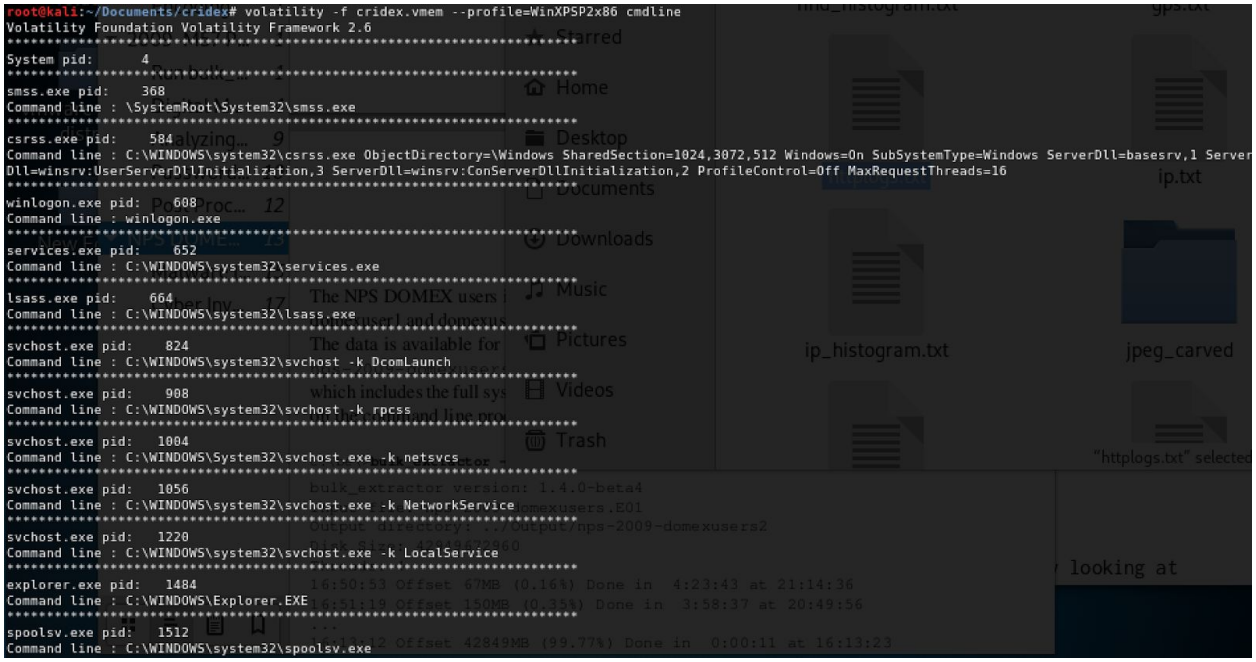
Step 7: Now using the “sockets” command, we can see a list of open sockets that allow the connections. Notice that the previously identified process 1484 has opened a connection on port 1038 which matches up with the previous command’s detail identification. However, there is also a connection on port 1037, but the system does not identify any sockets open on port 1037 nor attached to the process id of 1484. This is highly suspicious as you can not establish a connection without an open port.

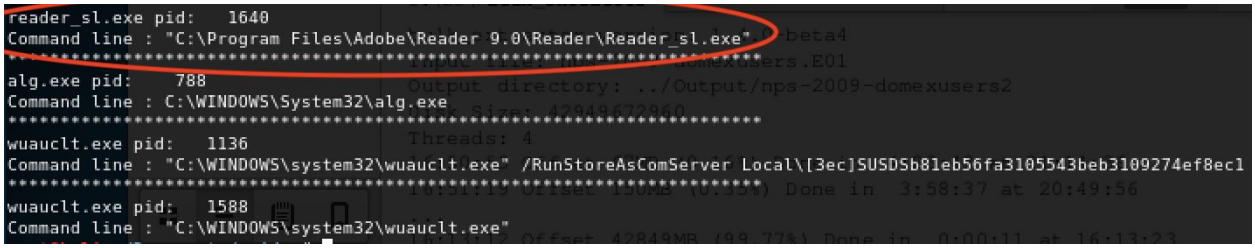
```
root@kali:~/Documents/cridex# volatility -f cridex.vmem --profile=WinXPSP2x86 sockets
Volatility Foundation Volatility Framework 2.6
Offset(V)       PID   Port  Proto Protocol        Address         Create Time
---------- -------- ------ ------ -------------- --------------- -----------
0x81ddb780      664    500     17 UDP            0.0.0.0         2012-07-22 02:42:53 UTC+0000
0x82240d08     1484   1038      6 TCP            0.0.0.0         2012-07-22 02:44:45 UTC+0000
0x81dd7618     1220   1900     17 UDP            172.16.112.128  2012-07-22 02:43:01 UTC+0000
0x82125610      788   1028      6 TCP            127.0.0.1       2012-07-22 02:43:01 UTC+0000
0x8219cc08        4    445      6 TCP            0.0.0.0         2012-07-22 02:42:31 UTC+0000
0x81ec23b0      908    135      6 TCP            0.0.0.0         2012-07-22 02:42:33 UTC+0000
0x82276878        4    139      6 TCP            172.16.112.128  2012-07-22 02:42:38 UTC+0000
0x82277460        4    137     17 UDP            172.16.112.128  2012-07-22 02:42:38 UTC+0000
0x81e76620     1004    123     17 UDP            127.0.0.1       2012-07-22 02:43:01 UTC+0000
0x82172808      664      0    255 Reserved       0.0.0.0         2012-07-22 02:42:53 UTC+0000
0x81e3f460        4    138     17 UDP            172.16.112.128  2012-07-22 02:42:38 UTC+0000
0x821f0630     1004    123     17 UDP            172.16.112.128  2012-07-22 02:43:01 UTC+0000
0x822cd2b0     1220   1900     17 UDP            127.0.0.1       2012-07-22 02:43:01 UTC+0000
0x82172c50      664   4500     17 UDP            0.0.0.0         2012-07-22 02:42:53 UTC+0000
0x821f0d00        4    445     17 UDP            0.0.0.0         2012-07-22 02:42:31 UTC+0000
```

```
root@kali:~/Documents/cridex# volatility -f cridex.vmem --profile=WinXPSP2x86 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
```

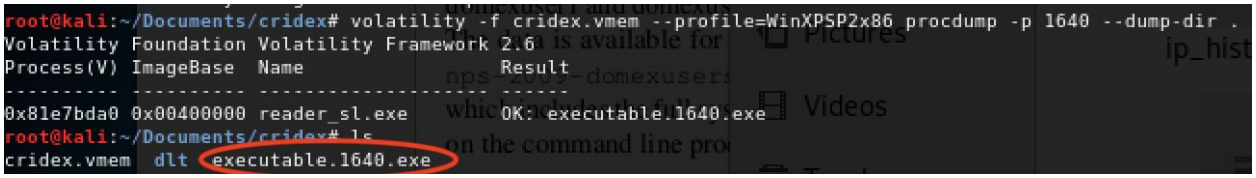
Step 8: Another thing we can look into is the commands run on the system by running “cmdline”. Circled in red we can see another form of suspicious activity. It appears that the adobe reader process that we have been following closely was spawned via command line. This is suspicious as there are not many people who would open the adobe reader application via

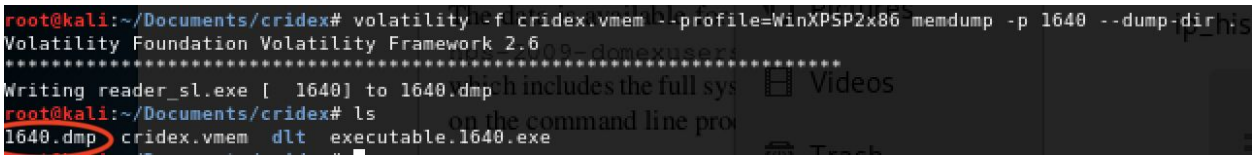
command line. Furthermore, this is inconsistent with our previous discovery that adobe reader was opened by file explorer which implies a person has double clicked a file whilst browsing files to open it with the adobe reader application.



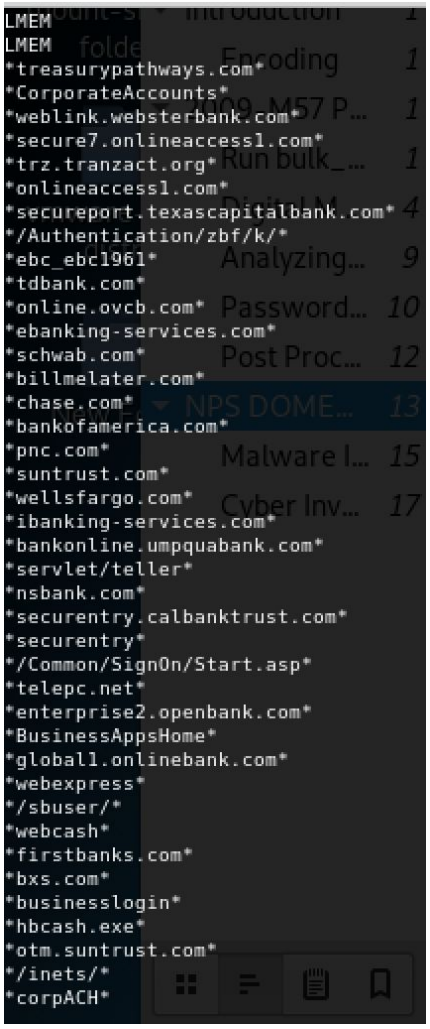


Step 9: Now that we have an idea of where all the suspicious activity is originating from, we can use procdump to dump a process' executable (which in this case will be the malware) and memdump to extract all memory resident pages in a process.



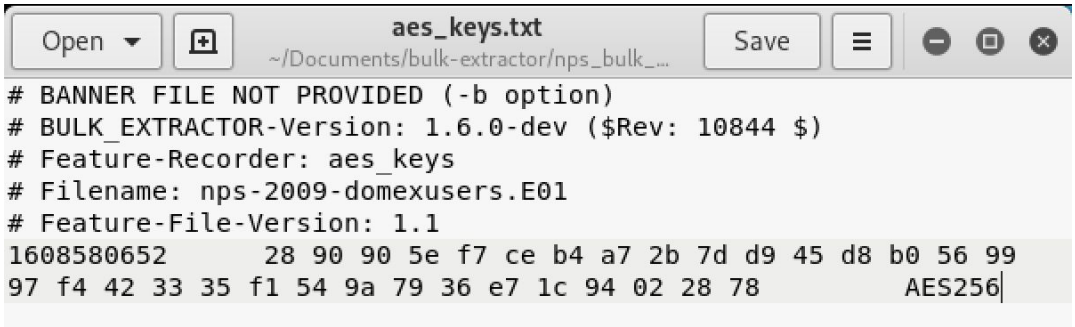


Step 10: Using strings, we can pull out a large portion of important human-readable information within the memory dump. There are a whole bunch of bank urls which lines up with the case where this was a banking malware that took details regarding user's banking details.



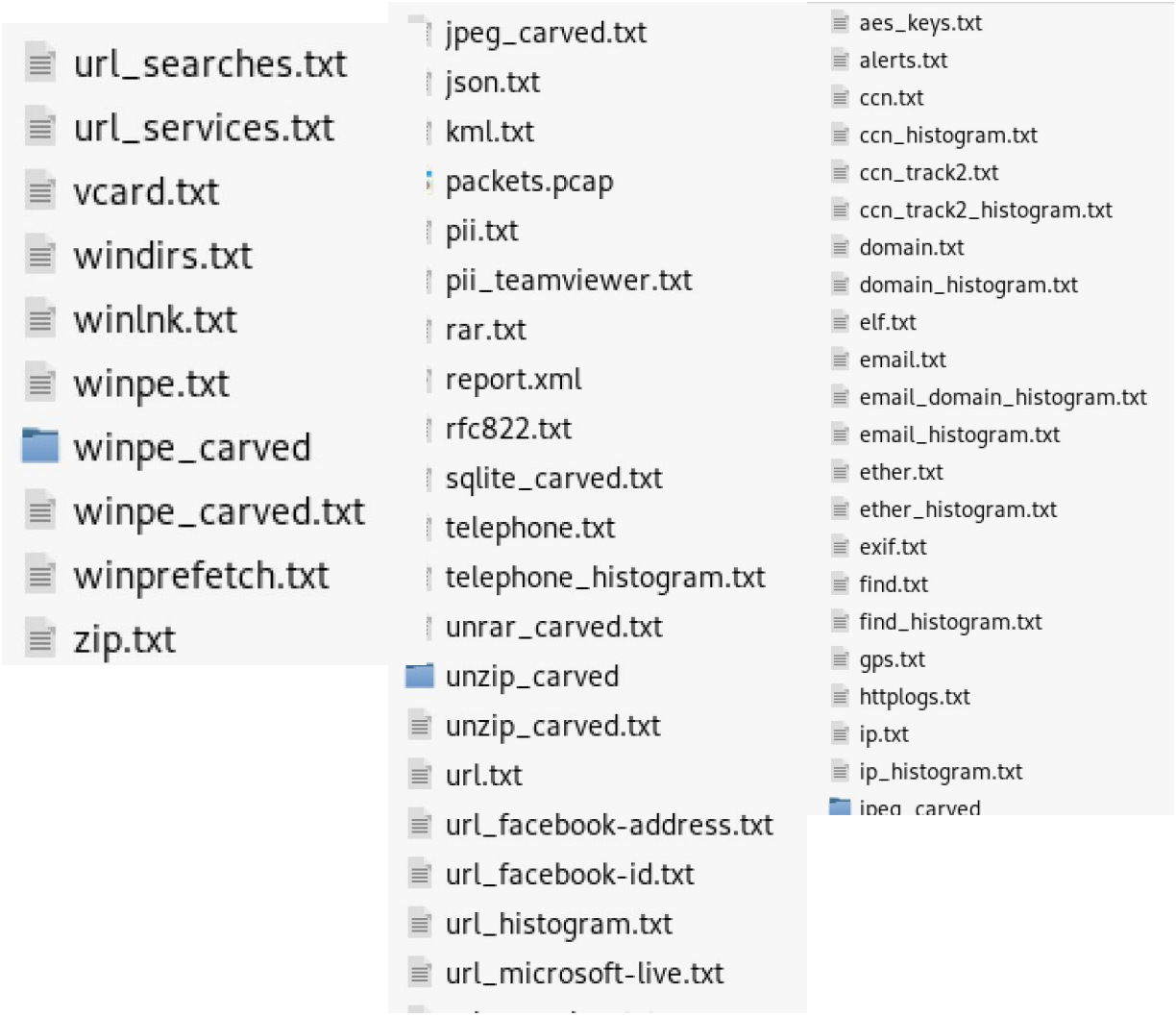
[PART 1]

Output Text Files for the AES key as was extracted by bulk_extractor into aes_keys.txt

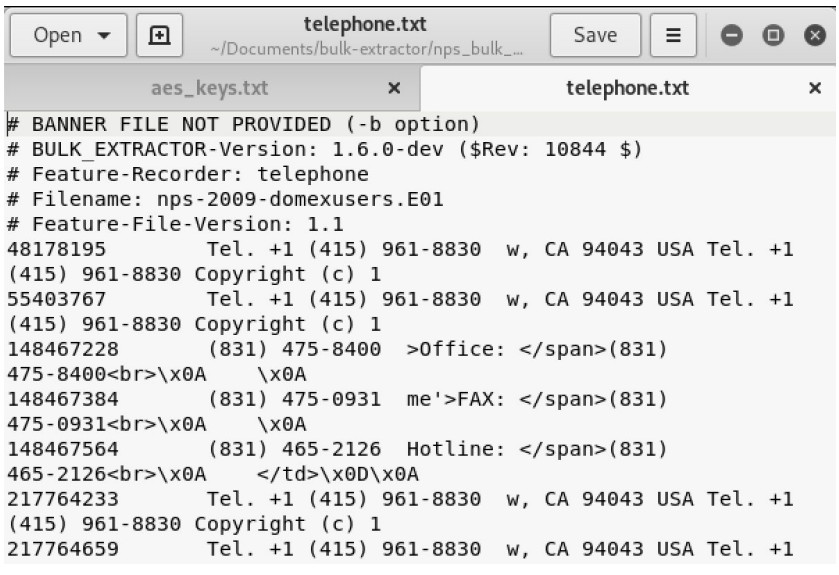


Evidence of other bulk_extractor artefacts

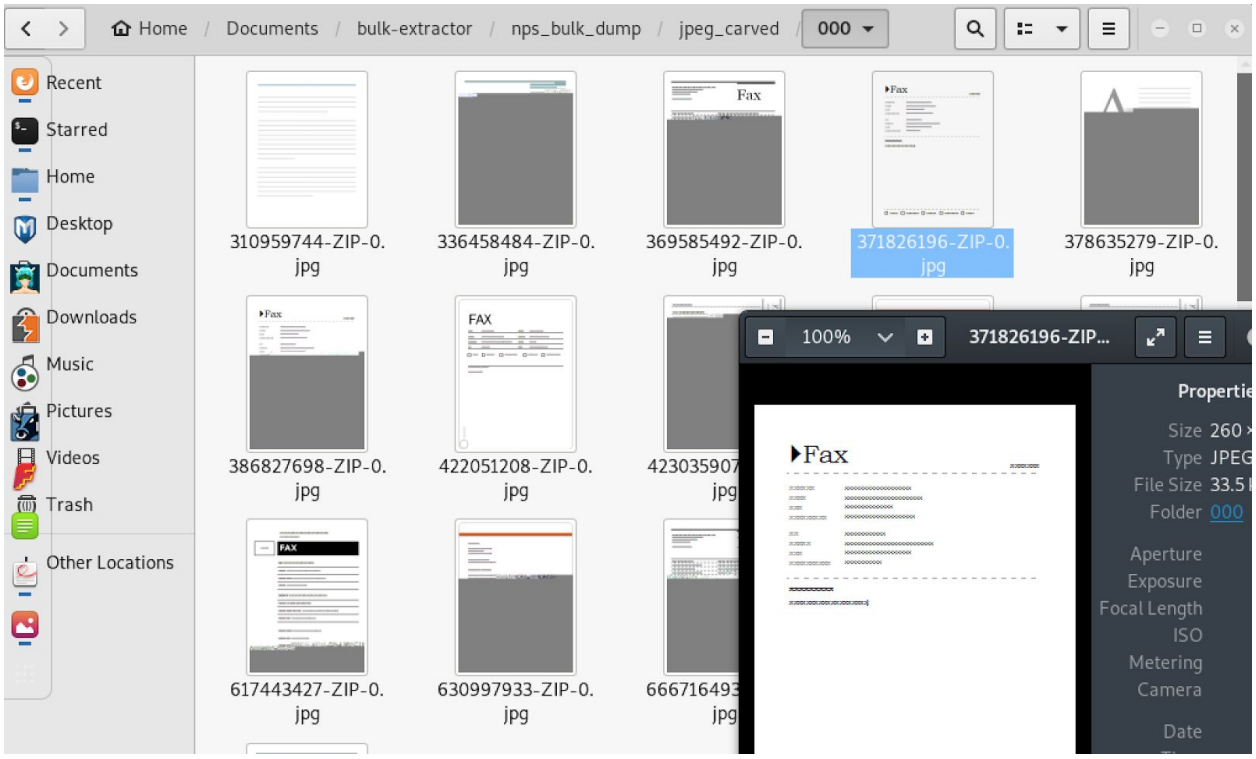
As can be seen in the photos below, bulk_extractor generate many artefacts.



Hence, I will only show evidence of a few artefacts generated.
Bulk extractor was able to identify a bunch of telephone/fax/contact numbers. This is an important file as it may indicate who the user of the machine was in contact with.



There are also a bunch of images extracted. Although some are damaged, it is evident that there are images which may be financial statements, or other key pieces of information that may aid the expert's opinion.



The url extraction is also interesting as it may indicate what websites the user was searching or has visited. There is also a packets.pcap that when opened in wireshark indicates some of the traffic from the device.