Cridex Memory Analysis

By: Hridaya Patel

Part 1: Image Identification

I first installed the Volatility memory forensics tool. It can be found at: `https://www.volatilityfoundation.org/releases`. You can also you pre-loaded VM's such as Kali Linux or SIFT.

We have our memory sample and now we have to identify what kind of image it is. We can use the *imageinfo* plugin to get information on our sample.

```
root@kali:~/Downloads# 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/Downloads/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
```

This info gives us the start we need when we do further analysis using different plugins. We will use the parameter --profile=PROFILE. In our case it will be *--profile=WinXPSP2x86*.

Note: *imageinfo* will give you *suggestions* as to what the profile might be.

Part 2: Identify Rogue Processes

There are several plugins in Volatility that allow us to see the processes that are running. Some of these include: psscan, pstree, pslist, psxview.

We can see what processes were previously terminated and processes that have been unlinked by a rootkit by using *psscan.*

```
root@kali:~/Downloads# volatility -f cridex.vmem --profile=WinXPSP2x86 psscan
Volatility Foundation Volatility Framework 2.6
Offset(P)          Name                PID   PPID PDB        Time created                   Time exited
------------------ ---------------- ------ ------ ---------- ------------------------------ ------------------------------
0x0000000002029ab8 svchost.exe         908    652 0x079400e0 2012-07-22 02:42:33 UTC+0000
0x000000000202a3b8 lsass.exe           664    608 0x079400a0 2012-07-22 02:42:32 UTC+0000
0x000000000202ab28 services.exe        652    608 0x07940080 2012-07-22 02:42:32 UTC+0000
0x000000000207bda0 reader_sl.exe      1640   1484 0x079401e0 2012-07-22 02:42:36 UTC+0000
0x00000000020b17b8 spoolsv.exe        1512    652 0x079401c0 2012-07-22 02:42:36 UTC+0000
0x000000000225bda0 wuauclt.exe        1588   1004 0x07940200 2012-07-22 02:44:01 UTC+0000
0x00000000022e8da0 alg.exe             788    652 0x07940140 2012-07-22 02:43:01 UTC+0000
0x00000000023dea70 explorer.exe       1484   1464 0x079401a0 2012-07-22 02:42:36 UTC+0000
0x00000000023dfda0 svchost.exe        1056    652 0x07940120 2012-07-22 02:42:33 UTC+0000
0x00000000023fcda0 wuauclt.exe        1136   1004 0x07940180 2012-07-22 02:43:46 UTC+0000
0x0000000002495650 svchost.exe        1220    652 0x07940160 2012-07-22 02:42:35 UTC+0000
0x0000000002498700 winlogon.exe        608    368 0x07940060 2012-07-22 02:42:32 UTC+0000
0x00000000024a0598 csrss.exe           584    368 0x07940040 2012-07-22 02:42:32 UTC+0000
0x00000000024f1020 smss.exe            368      4 0x07940020 2012-07-22 02:42:31 UTC+0000
0x00000000025001d0 svchost.exe        1004    652 0x07940100 2012-07-22 02:42:33 UTC+0000
0x0000000002511360 svchost.exe         824    652 0x079400c0 2012-07-22 02:42:33 UTC+0000
0x00000000025c89c8 System                4      0 0x002fe000
```

The pslist plugin shows the offset, process name, PID, PPID, # of threads, # of handles, and date/time.

pstree is an alternate to pslist and can also be used:

```
root@kali:~/Downloads# 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
```

From this I noticed an odd process that was one of the last processes running named "reader_sl.exe" with the parent process (PPID) being "explorer.exe". Before going further, I ran the psxview command which cross reference processes with various lists.

```
root@kali:~/Downloads# 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
```

So, no processes seem to be hidden. If there were processes being hidden we would have seen "False" in the the first two columns (**pslist** and **psscan**).

I wanted to see what the open connections on the computer. We can run plugins such as connscan, and sockets.

```
root@kali:~/Downloads# 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:~/Downloads# 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
```

Connscan is a scanner for TCP connections and sockets will detect listening sockets for any protocol (TCP, RAW, UDP, etc).

Immediately we are able to see that two TCP connections are being used by the PID 1484 and one of the connections is still open communicating with IP: 41.168.5.140 via port 1038.

I want to take a look at the last commands run on the system so I try using cmdscan and consoles which returned nothing. I also tried cmdline which gave me some interesting information.

```
root@kali:~/Downloads# volatility -f cridex.vmem --profile=WinXPSP2x86 cmdline
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=1024,3072,512 Windows=On SubSystemType=Windows S
erverDll=basesrv,1 ServerDll=winsrv:UserServerDllInitialization,3 ServerDll=winsrv:ConServerDllInitialization,2 ProfileControl=Off M
axRequestThreads=16
************************************************************************
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]SUSDSb81eb56fa3105543beb3109274ef8ec1
************************************************************************
wuauclt.exe pid:   1588
Command line : "C:\WINDOWS\system32\wuauclt.exe"
```

From these cmdline arguments the file “reader_sl.exe” is really suspicious. It seems to reside in Adobe\Reader 9.0\reader_sl.exe in the C drive of the windows file system. PID 1640 was launched as part of the explorer process and is an Adobe Reader application. But we were able to see an external IP (41.168.5.14:1038) used by the very same process….

We can dig further using the memdump and procdump plugins. Memdump extracts all memory resident pages in a process into an individual file. We can use *-D or –dump-dir=DIR*.

```
root@kali:~/Downloads# volatility -f cridex.vmem --profile=WinXPSP2x86 procdump -p 1640 --dump-dir .
Volatility Foundation Volatility Framework 2.6
Process(V) ImageBase  Name                 Result
---------- ---------- -------------------- ------
0x81e7bda0 0x00400000 reader_sl.exe        OK: executable.1640.exe
root@kali:~/Downloads# volatility -f cridex.vmem --profile=WinXPSP2x86 procdump -p 1056 --dump-dir .
Volatility Foundation Volatility Framework 2.6
Process(V) ImageBase  Name                 Result
---------- ---------- -------------------- ------
0x821dfda0 0x01000000 svchost.exe          OK: executable.1056.exe
root@kali:~/Downloads# volatility -f cridex.vmem --profile=WinXPSP2x86 memdump -p 1640 --dump-dir .
Volatility Foundation Volatility Framework 2.6
************************************************************************
Writing reader_sl.exe [  1640] to 1640.dmp
root@kali:~/Downloads# volatility -f cridex.vmem --profile=WinXPSP2x86 memdump -p 1056 --dump-dir .
Volatility Foundation Volatility Framework 2.6
************************************************************************
Writing svchost.exe [  1056] to 1056.dmp
```

***I meant to dump PID 1484 NOT 1056

The first file “executable.1640.exe” is a procurement of the executable “Reader_sl.exe” and the dump extracted “1640.dmp” represents the addressable memory of the processes.

I analyzed these commands using strings and piped it with grep in order to find a correlation with the IP 41.168.5.140. The grep and -C #NUMBER command to get the previous and next lines.

The executable “Reader_sl.exe” is communicating with the IP 41.168.5.140 using POST requests.

If you run strings and pipe it with less (strings 1640.dmp | less) we can see the file. I found a significant amount of banking and financial domains…

Part 3: Hunting for Malicious Activity

Anomalous activity from two processes (explorer.exe and reader_sl.exe) suggests that there might be some code injection. I used the malfind plugin to confirm this theory.

```
root@kali:~/Downloads# volatility -f cridex.vmem --profile=WinXPSP2x86 malfind -p 1484
Volatility Foundation Volatility Framework 2.6
Process: explorer.exe Pid: 1484 Address: 0x1460000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 33, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x01460000  4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00   MZ..............
0x01460010  b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00   ........@.......
0x01460020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x01460030  00 00 00 00 00 00 00 00 00 00 00 00 e0 00 00 00   ................
```

```
root@kali:~/Downloads# volatility -f cridex.vmem --profile=WinXPSP2x86 malfind -p 1640
Volatility Foundation Volatility Framework 2.6
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   ................
```

I found the virtual address space and dumped the data using vaddump for the two processes. After I had this .dmp I was able to confirm that memory was allocated in the targeted processe(s) with the VirtualAllocEx, the injected code was written to the targeted processes via WriteProcessMemory, and malicious thread of execution was spawned via CreateRemoteThread. It seems to be that explorer.exe and reader_sl.exe seem to be injected with the same code.

Using the apihooks plugin I was able to see what kind of .dll hook functions (inline) were being injected into the two processes in question (reader_sl.exe and explorer.exe). The functions hooked ***ws_32.dll*** are pretty much all sock related (probably going over port 8080 like we observed in our socket and connscan), ***secure32.dll*** functions are related to encrypting/decrypting data in transit. The ***ntdll.dll*** functions seem to be used for loading a module and resume the thread of the execution of the injected code.

```
Hook mode: Usermode
Hook type: Inline/Trampoline
Process: 1640 (reader_sl.exe)
Victim module: Secur32.dll (0x77fe0000 - 0x77ff1000)
Function: Secur32.dll!DecryptMessage at 0x77fea64a
Hook address: 0x3d9f20
Hooking module: <unknown>
```

```
Hook mode: Usermode
Hook type: Inline/Trampoline
Process: 1640 (reader_sl.exe)
Victim module: WS2_32.dll (0x71ab0000 - 0x71ac7000)
Function: WS2_32.dll!WSARecv at 0x71ab4cb5
Hook address: 0x3d6dd0
Hooking module: <unknown>
```

```
Hook mode: Usermode
Hook type: Inline/Trampoline
Process: 1640 (reader_sl.exe)
Victim module: ntdll.dll (0x7c900000 - 0x7c9af000)
Function: ntdll.dll!LdrLoadDll at 0x7c9163a3
Hook address: 0x3da300
Hooking module: <unknown>
```

```
Hook mode: Usermode
Hook type: Inline/Trampoline
Process: 1640 (reader_sl.exe)
Victim module: ntdll.dll (0x7c900000 - 0x7c9af000)
Function: ntdll.dll!NtResumeThread at 0x7c90db20
Hook address: 0x3da330
Hooking module: <unknown>
```

```
Hook mode: Usermode
Hook type: Inline/Trampoline
Process: 1640 (reader_sl.exe)
Victim module: Secur32.dll (0x77fe0000 - 0x77ff1000)
Function: Secur32.dll!EncryptMessage at 0x77fea5fb
Hook address: 0x3da1e0
Hooking module: <unknown>
```

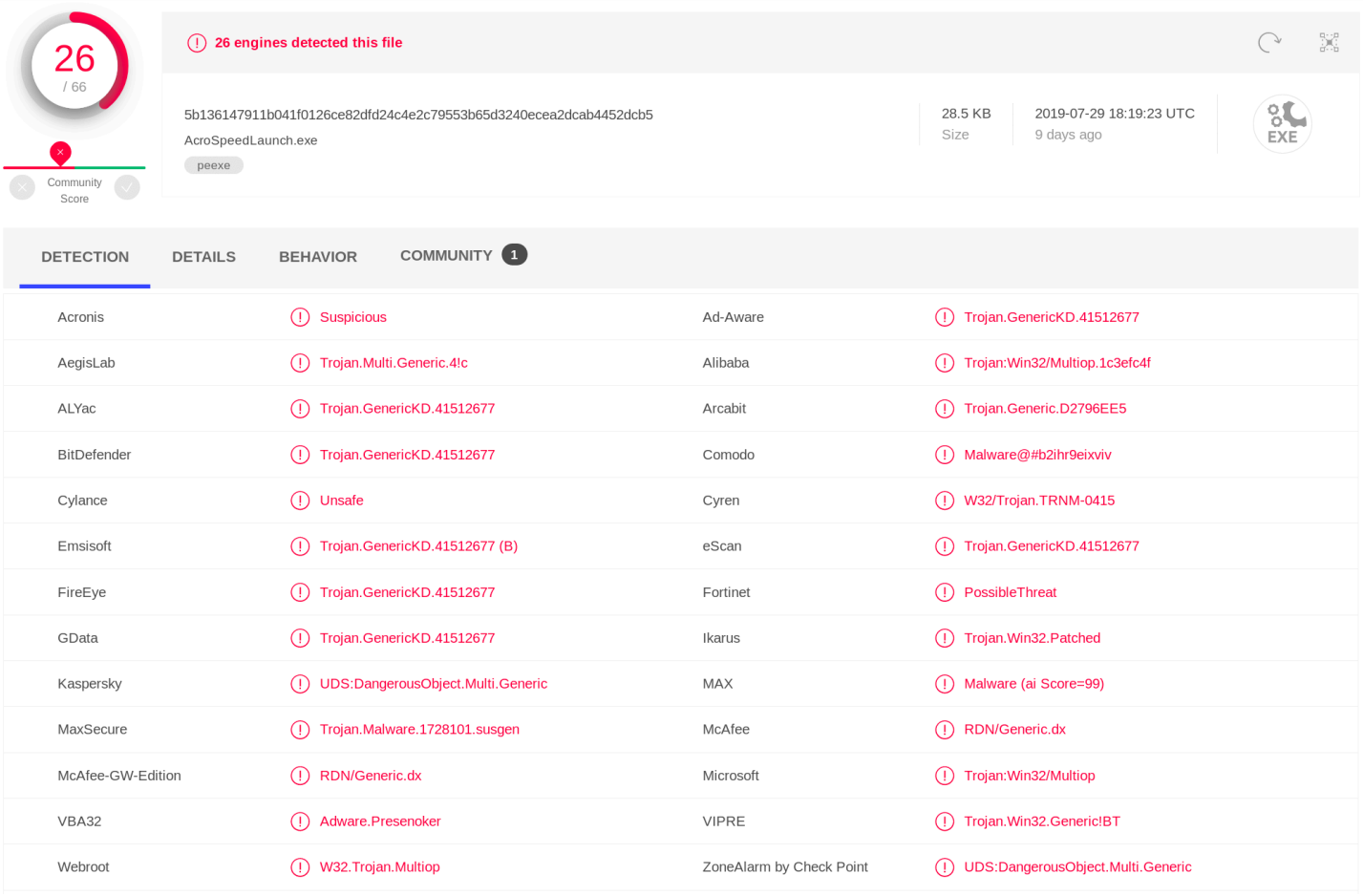
After taking a look at the domains that “Reader_sl.exe” is communicating with I decided this was likely an IOC and to dig deeper and run some dynamic analysis. I am not currently as capable as I’d like to be with reverse analysis so I decided to use VT and see what kind of results I would get.

List of domains found in 1640.dmp (not complete)

chase.com *bankofamerica.com* *pnc.com* *suntrust.com* *wellsfargo.com* *ibanking-services.com* *bankonline.umpquabank.com* *nsbank.com* *securentry.calbanktrust.com* *enterprise2.openbank.com* *BusinessAppsHome* *global1.onlinebank.com* *webexpress* *webcash* *firstbanks.com* *bxs.com* *businesslogin* *otm.suntrust.com* *fnfgbusinessonline.enterprisebanker.com* *lakecitybank.webcashmgmt.com* *bob.sovereignbank.com* *directline4biz.com* *e-moneyger.com* *securentrycorp.amegybank.com* *treasurypathways.com* *weblink.websterbank.com* *secure7.onlineaccess1.com* *trz.tranzact.org* *onlineaccess1.com* *secureport.texascapitalbank.com* *ebc_ebc1961* *tdbank.com* *online.ovcb.com* *ebanking-services.com* *schwab.com* *billmelater.com* *chase.com* *bankofamerica.com* *pnc.com* *suntrust.com* !!!! *wellsfargo.com* *ibanking-services.com* *bankonline.umpquabank.com* *nsbank.com* *securentry.calbanktrust.com* *telepc.net* *enterprise2.openbank.com* *global1.onlinebank.com* *firstbanks.com* *bxs.com* *hsbc.co.uk* *ablv.com* *accessbankplc.com* *alphabank.com.cy* *baltikums.com* *baltikums.eu* *banesco.com.pa* *bankaustria.at* *banknet.lv* *bankofcyprus.com* *bobibanking.com* *butterfieldonline.ky* *cimbanque.net*	*suntrust.com* *wellsfargo.com* *ibanking-services.com* *bankonline.umpquabank.com* *nsbank.com* *securentry.calbanktrust.com* *telepc.net* *enterprise2.openbank.com* *global1.onlinebank.com* *firstbanks.com* *bxs.com* *hsbc.co.uk* *ablv.com* *accessbankplc.com* *alphabank.com.cy* *baltikums.com* *baltikums.eu* *banesco.com.pa* *bankaustria.at* *banknet.lv* *bankofcyprus.com* *bobibanking.com* *butterfieldonline.ky* *cimbanque.net* *cs.directnet.com* *directnet.com* *danskebanka.lv* *ebank.laiki.com* *ebank.rcbcy.com* *ebemo.bemobank.com* *e-norvik.lv* *eurobankefg.com* *eurobankefg.com.cy* *ekp.lv* *fbmedirect.com* *hblibank.com* *hellenicnetbanking.com* *hiponet.lv* *hkbea* *ibanka.seb.lv* *ib.baltikums.com* *geonline.lv* *ib.dnb.lv* *bib.lv* *ib.snoras.com* *i-linija.lt* *loyalbank.com* *marfinbank.com.cy* *multinetbank.eu* *nordea.com* *secure.ltbbank.com* *secure.ltblv.com* *swedbank.lv* *norvik.lv* *online.alphabank.com.cy* *online.citadele.lv* *online.lkb.lv* *cs.directnet.com*

VirusTotal Analysis:

https://www.virustotal.com/gui/file/5b136147911b041f0126ce82dfd24c4e2c79553b65d3240ecea2dcab4452dcb5/details

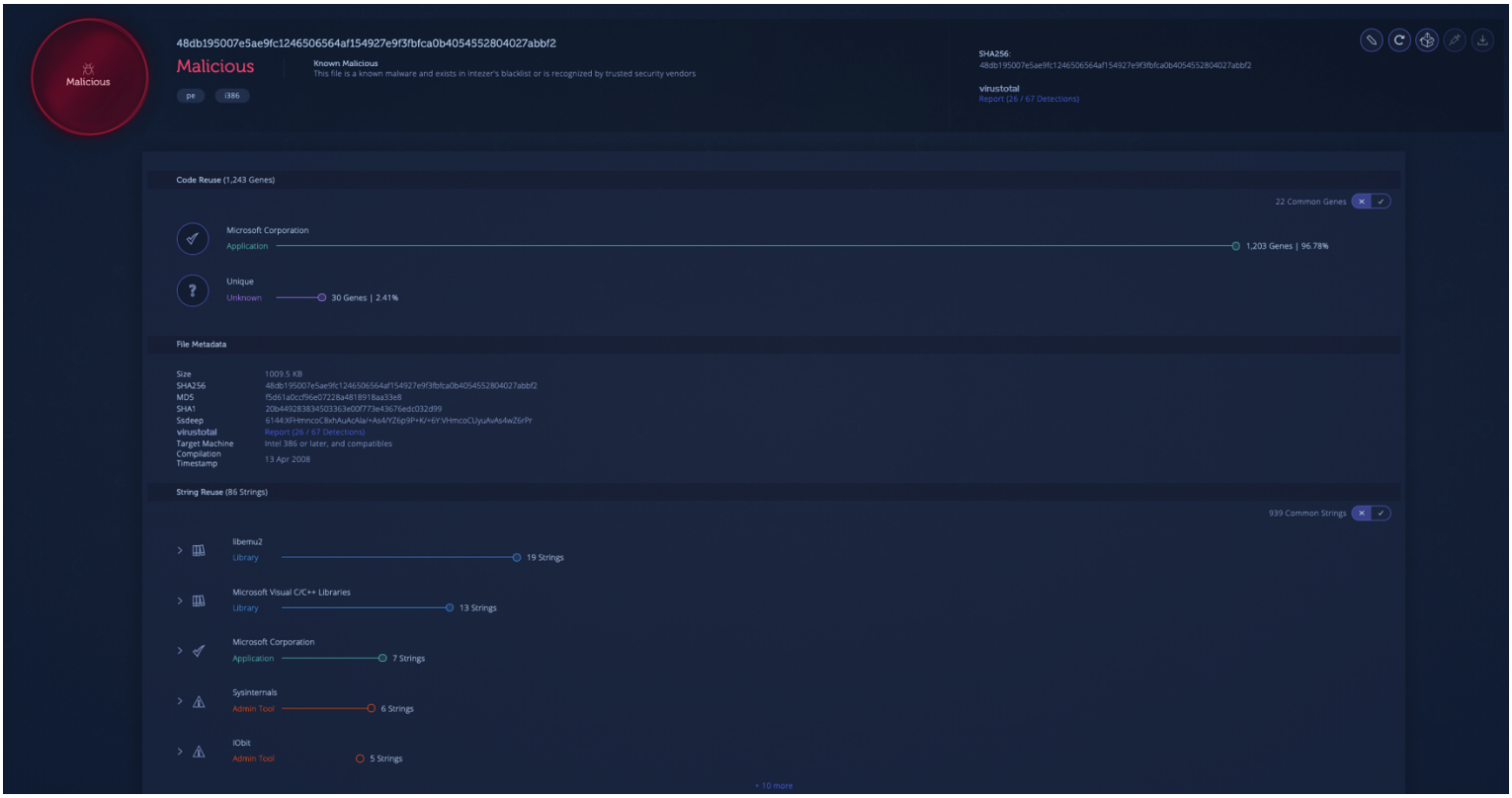


I've included the link to the VT report if anyone wants to investigate further and see what kind of details are in the VT analysis.

Next step would be to take a look at the encrypted messages being communicated and also see what kind of data is being exfil from the system. Pretty much the only way I can think of is to reverse engineer the payload that was taken out of the memory regions of PIDs 1640 and 1484 (explorer.exe and reader_sl.exe).

Intezer Analysis:

https://analyze.intezer.com/#/analyses/66042113-c83f-4276-ae24-0c43e4374204



<u>Cridex Analysis Takeaways:</u>

System: Windows XP

Malware: Cridex

Attack Technique: Code Injection -> Exfiltration

IOC's I found:

- explorer.exe
- reader_sl.exe
- 41.168.5[.]140:8080
- 125.19.103[.]198:8080
- MD5 12cf6583f5a9171a1d621ae02b4eb626
- SHA-1 61ed2778d7669d6835823369fd04278626303362
- SHA-256 5b136147911b041f0126ce82dfd24c4e2c79553b65d3240ecea2dcab4452dcb5

Notes:

- creates process
- collects exfiltrates data
- communication is encrypted when going out
- is able to traverse the file system and achieve some kind of persistence
- contacts external IP's/hosts

- injected two benign processes with payload

How Can We Analyze Further?

- grep VAD dump for IP's relating to PIDs 1640 and 1484
    - create graph in Maltego to correlate the IP's with country, designation, DNS, and ASN
- perform reverse analysis of payload in order to better understand what has been taken/is being taken
- To better understand the obfuscated strings we can use a tool from FireEye called Floss
- There is also Rekall made by Google that can also be used to understand Windows memory samples

Part 4: Remediation

Prevention:

- Stop using Windows XP in 2019 ☺️
- Search in the process dump file for the IP's listed as IOC's
- Analyze each possible infected host to see if it is linked to the malware
- Train employees not to open attachments from suspicious emails

Deletion and Containment

- Use an antivirus that can detect this malware and has a signature in place to catch and delete it.
- Isolate the systems quickly to prevent further spreading

*** *I might keep adding to this report as I learn more analysis techniques and become a more comfortable with reverse analysis* ***

Sources:

- https://github.com/volatilityfoundation/volatility/wiki
- https://github.com/volatilityfoundation/volatility/wiki/Command-Reference
- https://github.com/volatilityfoundation/volatility/wiki/Command-Reference-Mal
- https://digital-forensics.sans.org/media/volatility-memory-forensics-cheat-sheet.pdf
- https://www.youtube.com/watch?v=BMFCdAGxVN4

- Kali Linux and SIFT VM's

- James Quinn (SOC Analyst at Binary Defense): Helped me understand certain concepts and explain to me what I should be looking for, what kind of process to follow, and general windows tips. Also peaked my interest in setting up my own Honeypot lab to see network traffic and collect wild malware samples.

- *The Art of Memory Forensics* by AAron Walters, Andrew Case, Jamie Levy, and Michael Hale Ligh

- Andrea Fortuna for publishing excellent content online about Volatility usage and analyzing malware/memory