ECE 570: Project Report 1

Date: June 4th, 2024

By: Payton Murdoch, V00904677

&

Yun Ma, V01018599

Table of Contents

Foreword

Before we delve into the project content. It is important to note that Yun Ma will be utilizing a Kali machine with Volatility 3 installed within it, while Payton Murdoch will be utilizing the Kali machine provided for the course loaded with Volatility 2.6. Therefore, the screenshots for various tasks will depict slightly different versions with different layouts.

1. Identify Suspicious running processes.

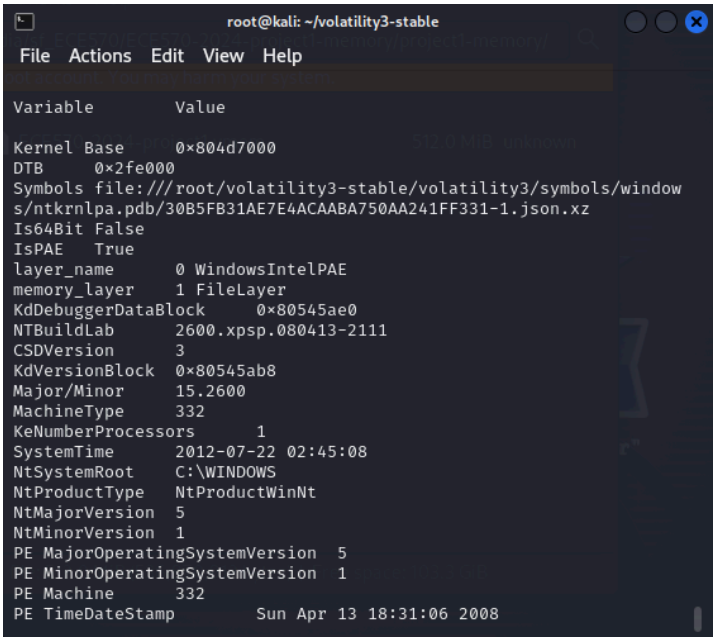


Figure 1: Meminfo scan for project1.vmem file.

The memory dump in Figure 1 depicts a Windows XP Professional Service Pack 3 machine on a 32-bit architecture. Physical Address Extension (PAE) allows the system to use over 4 GB of physical memory on a 32-bit OS. The kernel base address is 0x804d7000, and the Directory Table Base (DTB) is 0x2fe000. The system time in the dump is set to July 22, 2012.

Next, we will utilize the command volatility command pslist to depict and show the running processes within this dump file, as shown in Figure 2.

PID	PPID	ImageFileName	Offset(V)	Threads	Handles	SessionId	Wow64	CreateTime	ExitTime	File output
4	0	System	0x823c89c8	53	240	N/A	False	N/A	N/A	Disabled
368	4	smss.exe	0x822f1020	3	19	N/A	False	2012-07-22 02:42:31.000000	N/A	Disabled
584	368	csrss.exe	0x822a0598	9	326	0	False	2012-07-22 02:42:32.000000	N/A	Disabled
608	368	winlogon.exe	0x82298700	23	519	0	False	2012-07-22 02:42:32.000000	N/A	Disabled
652	608	services.exe	0x81e2ab28	16	243	0	False	2012-07-22 02:42:32.000000	N/A	Disabled
664	608	lsass.exe	0x81e2a3b8	24	330	0	False	2012-07-22 02:42:32.000000	N/A	Disabled
824	652	svchost.exe	0x82311360	20	194	0	False	2012-07-22 02:42:33.000000	N/A	Disabled
908	652	svchost.exe	0x81e29ab8	9	226	0	False	2012-07-22 02:42:33.000000	N/A	Disabled
1004	652	svchost.exe	0x823001d0	64	1118	0	False	2012-07-22 02:42:33.000000	N/A	Disabled
1056	652	svchost.exe	0x821dfda0	5	60	0	False	2012-07-22 02:42:33.000000	N/A	Disabled
1220	652	svchost.exe	0x82295650	15	197	0	False	2012-07-22 02:42:35.000000	N/A	Disabled
1484	1464	explorer.exe	0x821dea70	17	415	0	False	2012-07-22 02:42:36.000000	N/A	Disabled
1512	652	spoolsv.exe	0x81eb17b8	14	113	0	False	2012-07-22 02:42:36.000000	N/A	Disabled
1640	1484	reader_sl.exe	0x81e7bda0	5	39	0	False	2012-07-22 02:42:36.000000	N/A	Disabled
788	652	alg.exe	0x820e8da0	7	104	0	False	2012-07-22 02:43:01.000000	N/A	Disabled
1136	1004	wuauclt.exe	0x821fcda0	8	173	0	False	2012-07-22 02:43:46.000000	N/A	Disabled
1588	1004	wuauclt.exe	0x8205bda0	5	132	0	False	2012-07-22 02:44:01.000000	N/A	Disabled

Figure 2: Process list for vmem dump file.

We would like to highlight a few processes that catch our attention. First, there is a high diversity of svchost.exe files. This is not an indicator of suspicious activity; however, we would like to explore these

closer as it can be easy to camouflage suspicious processes amongst them. The most prominent oddity comes from the process with PID 1004, as its number of threads and handles greatly exceeds those of other svchost.exe files. With this in mind, we can look further into the process with the getsids command, as shown in Figure 3.

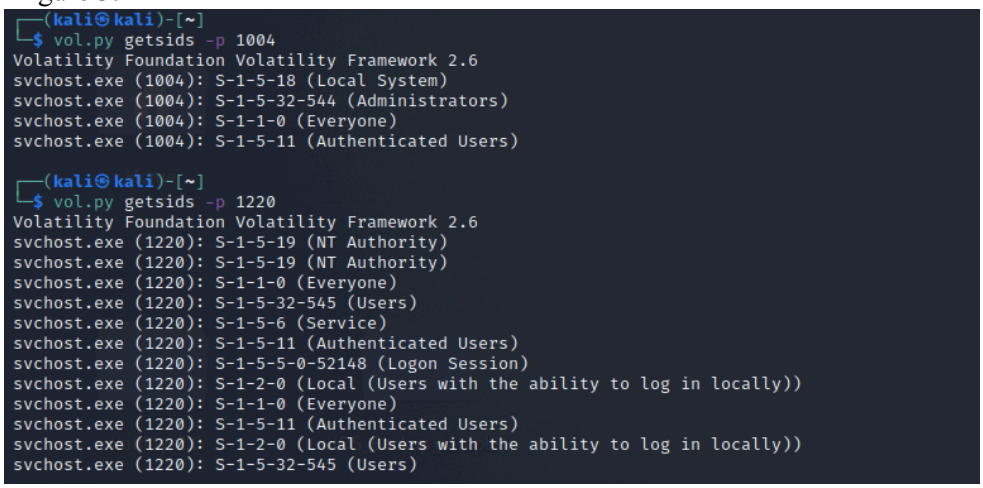


Figure 3: SID comparison between two svchost.exe processes.

Figure 3 shows a large discrepancy between the SIDs for PID 1004 and a normally spawned PID 1220 svchost.exe file, indicating that this process may cause malicious actions as it contains Admin SIDs, which are uncommon. Next, another peculiar process, PID 1484 explorer.exe, is a parent process for reader_sl.exe PID 1640. These processes arouse suspicion they spawn from a parent PID, which does not exist as there is no PID 1464 in our running process table. We observe the SIDs in Figure 4 and note Admin SIDs in this process, which is suspicious and not a common practice.

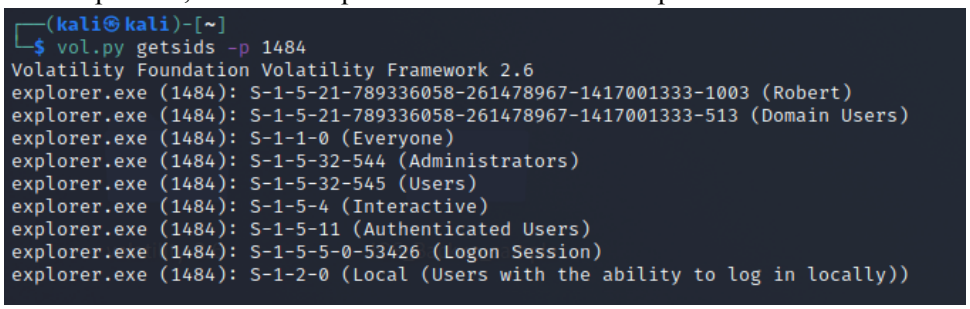


Figure 4: SID of explorer.exe process.

Lastly, we want to address the wuauclt.exe process running. This is the Windows Update Agent Utility process on a Windows machine, and it is peculiar that multiple are running at the given moment. Hence, we examined the SIDs of the processes, as shown in Figure 5.

```
┌──(kali㉿kali)-[~]
└─$ vol.py getsids -p 1136
Volatility Foundation Volatility Framework 2.6
wuauclt.exe (1136): S-1-5-18 (Local System)
wuauclt.exe (1136): S-1-5-32-544 (Administrators)
wuauclt.exe (1136): S-1-1-0 (Everyone)
wuauclt.exe (1136): S-1-5-11 (Authenticated Users)

┌──(kali㉿kali)-[~]
└─$ vol.py getsids -p 1588
Volatility Foundation Volatility Framework 2.6
wuauclt.exe (1588): S-1-5-21-789336058-261478967-1417001333-1003 (Robert)
wuauclt.exe (1588): S-1-5-21-789336058-261478967-1417001333-513 (Domain Users)
wuauclt.exe (1588): S-1-1-0 (Everyone)
wuauclt.exe (1588): S-1-5-32-544 (Administrators)
wuauclt.exe (1588): S-1-5-32-545 (Users)
wuauclt.exe (1588): S-1-5-4 (Interactive)
wuauclt.exe (1588): S-1-5-11 (Authenticated Users)
wuauclt.exe (1588): S-1-5-5-0-53426 (Logon Session)
wuauclt.exe (1588): S-1-2-0 (Local (Users with the ability to log in locally))
```

Figure 5: Examination of wuauclt.exe processes.

As we see, the aforementioned figure contains multiple concerning instances. Both processes contain non-standard Administrator SIDs, matching those of previously noted suspicious processes. For example, PID 1136 matches the SIDs for PID 1004, and PID 1588 matches the SIDS for PID 1484, which could link the processes and cause us to question their legitimacy.

2. Determine and explain the relationships and identify the initial exploit.

From the processes noted above, We first consider where they come from to better understand the method responsible for the initial exploit. Figure 6, shown below, repeats the process list as a tree linking the parent and child processes with the pstree command.

```
┌──(kali㉿kali)-[~]
└─$ vol.py 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
```

Figure 6: PSTREE command for parent and child processes.

As we noted in the prior section, the suspicious processes we found were reader_sl.exe, explorer.exe, both instances of wuauclt.exe, and an instance of svchost.exe. Tracing these files back to their parents with the tree view of the processes is rather simple. Let us start with the svchost.exe with PID 1004, a part of the 5 svchost.exe processes that are children from the parent services.exe with PID 652. Services.exe spawns from winlogon.exe PID 368, which in turn spawns from parent smss.exe PID 4. Finally, smss.exe is a child process from the System with PID 0. Svchost.exe with PID 1004 is noted to be the parent of both wuauclt.exe processes, which further perpetuates our suspicion of these processes. Lastly, we note that reader_sl.exe spawns from the parent process explorer.exe. However, explorer.exe spawns from a parent

process which is no longer active. Through research, userinit.exe is the typical process that initializes explorer.exe and then quits.[1] Therefore, this is not suspicious; however, since the process was terminated, we cannot guarantee that it arose through legitimate means. To verify this, we also turn to Figure 7, noting the psxview for finding hidden processes and we see no confirmation of PID 1464 existing at the current moment.

```
┌──(kali㉿kali)-[~]
└─$ vol.py 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
```

Figure 7: Psxview of processes to see any hidden content.

We must speculate which process and method is responsible for the initial exploit. Utilizing volatility malfind functionality, we can inspect all processes for malicious activity. It gives us unique insight without specifying a PID, as shown in Figure 8.

```
┌──(kali㉿kali)-[~]
└─$ vol.py malfind
Volatility Foundation Volatility Framework 2.6
Process: csrss.exe Pid: 584 Address: 0x7f6f0000
Vad Tag: Vad  Protection: PAGE_EXECUTE_READWRITE
Flags: Protection: 6

0x7f6f0000  c8 00 00 00 91 01 00 00 ff ee ff ee 08 70 00 00   .............p..
0x7f6f0010  08 00 00 00 00 fe 00 00 00 00 10 00 00 20 00 00   ................
0x7f6f0020  00 02 00 00 00 20 00 00 8d 01 00 00 ff ef fd 7f   ................
0x7f6f0030  03 00 08 06 00 00 00 00 00 00 00 00 00 00 00 00   ................

Process: winlogon.exe Pid: 608 Address: 0x13410000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 4, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x13410000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x13410010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x13410020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x13410030  00 00 00 00 25 00 25 00 01 00 00 00 00 00 00 00   ....%.%.........

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   ................
```

```
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   ................
```

Figure 8: Malfind results with volatility.

As we can observe, malfind returns 4 processes containing malicious code, and each process supposedly runs on Windows machines by default. With this in mind, the initial exploit on this machine allowed for Process Hollowing to occur, replacing regions of legitimate processes with malicious code. Tracing the source of the malicious code using Figure 8 and the tree structure in Figure 7 allows us to speculate that winlogon.exe could be the initial process that has been hollowed. The code propagates, and hollows processes userinit.exe and then explorer.exe and reader_sl.exe, and this is supported by the PAGE_EXECUTE_READWRITE permissions noted by the malfind analysis.

3. From the suspicious processes, identify a hidden DLL.

We will investigate the explorer.exe process more closely. The first step is to utilize volatility's malfind functionality and specify it to only investigate pid 1484, as shown in Figure 9.

```
┌──(root㉿kali)-[~/volatility3-stable]
└─# python3 vol.py -f /media/sf_ECE570/ECE570-2024-project1-memory/project1-memory/ECE570-2024-project1.vmem windows.malfind --pid 1484
Volatility 3 Framework 2.5.2
WARNING  volatility3.framework.layers.vmware: No metadata file found alongside VMEM file. A VMSS or VMSN file may be required to correctly process a VMEM file
. These should be placed in the same directory with the same file name, e.g. ECE570-2024-project1.vmem and ECE570-2024-project1.vmss.
Progress:  100.00               PDB scanning finished
PID     Process Start VPN       End VPN Tag     Protection      CommitCharge    PrivateMemory   File output     Hexdump Disasm

1484    explorer.exe    0x1460000       0x1480fff       VadS    PAGE_EXECUTE_READWRITE  33      1       Disabled
4d 5a 90 00 03 00 00 00 MZ......
04 00 00 00 ff ff 00 00 ........
b8 00 00 00 00 00 00 00 ........
40 00 00 00 00 00 00 00 @.......
00 00 00 00 00 00 00 00 ........
00 00 00 00 00 00 00 00 ........
00 00 00 00 00 00 00 00 ........
00 00 00 00 e0 00 00 00 ........        4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00 b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0
0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 e0 00 00 00
```

Figure 9: Malfind on pid 1484.

We note that within the PID 1484 memory, there is notably malicious code containing an "MZ" signature, which indicates a Windows executable file, suggesting that this memory region contains executable code.[2] Furthermore, PAGE_EXECUTE_READWRITE indicates the memory region has both execution and write permissions. Examining this further, we delve into the dlllist for the process, as shown in Figure 10.

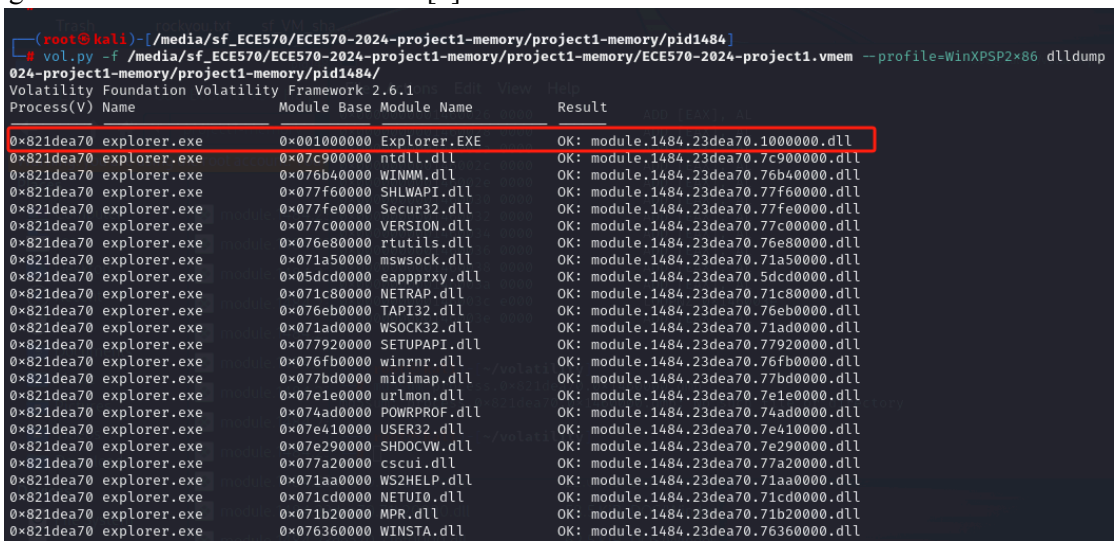
```
┌──(kali㉿kali)-[~]
└─$ vol.py dlllist -p 1484
Volatility Foundation Volatility Framework 2.6
************************************************************************
explorer.exe pid:   1484
Command line : C:\WINDOWS\Explorer.EXE
Service Pack 3

Base             Size  LoadCount Path
---------- ---------- ---------- ----
0x01000000    0xff000     0xffff C:\WINDOWS\Explorer.EXE
0x7c900000    0xaf000     0xffff C:\WINDOWS\system32\ntdll.dll
0x7c800000    0xf6000     0xffff C:\WINDOWS\system32\kernel32.dll
0x77dd0000    0x9b000     0xffff C:\WINDOWS\system32\ADVAPI32.dll
```

Figure 10: Dlllist for explorer.exe.

As we can see here, the command line for the explorer.exe file contains a peculiar capitalization not standard for an explorer.exe process. With this in mind, we can infer that this is an attempt to camouflage

the injected code through a hidden dll. If we were to examine further using dlldump, we could extract the dll of the associated process and paste the file through the VirusTotal website, as shown in Figures 11 and 12, to get confirmation of malicious code.[3]



```
┌──(root㉿kali)-[/media/sf_ECE570/ECE570-2024-project1-memory/project1-memory/pid1484]
└─# vol.py -f /media/sf_ECE570/ECE570-2024-project1-memory/project1-memory/ECE570-2024-project1.vmem --profile=WinXPSP2x86 dlldump
024-project1-memory/project1-memory/pid1484/
Volatility Foundation Volatility Framework 2.6.1
Process(V) Name                Module Base Module Name          Result
---------- ------------------- ----------- -------------------- ------
0x821dea70 explorer.exe        0x01000000 Explorer.EXE         OK: module.1484.23dea70.1000000.dll
0x821dea70 explorer.exe        0x7c900000 ntdll.dll            OK: module.1484.23dea70.7c900000.dll
0x821dea70 explorer.exe        0x76b40000 WINMM.dll            OK: module.1484.23dea70.76b40000.dll
0x821dea70 explorer.exe        0x77f60000 SHLWAPI.dll          OK: module.1484.23dea70.77f60000.dll
0x821dea70 explorer.exe        0x77fe0000 Secur32.dll          OK: module.1484.23dea70.77fe0000.dll
0x821dea70 explorer.exe        0x77c00000 VERSION.dll          OK: module.1484.23dea70.77c00000.dll
0x821dea70 explorer.exe        0x76e80000 rtutils.dll          OK: module.1484.23dea70.76e80000.dll
0x821dea70 explorer.exe        0x71a50000 mswsock.dll          OK: module.1484.23dea70.71a50000.dll
0x821dea70 explorer.exe        0x5dcd0000 eappprxy.dll         OK: module.1484.23dea70.5dcd0000.dll
0x821dea70 explorer.exe        0x71c80000 NETRAP.dll           OK: module.1484.23dea70.71c80000.dll
0x821dea70 explorer.exe        0x76eb0000 TAPI32.dll           OK: module.1484.23dea70.76eb0000.dll
0x821dea70 explorer.exe        0x71ad0000 WSOCK32.dll          OK: module.1484.23dea70.71ad0000.dll
0x821dea70 explorer.exe        0x77920000 SETUPAPI.dll         OK: module.1484.23dea70.77920000.dll
0x821dea70 explorer.exe        0x76fb0000 winrnr.dll           OK: module.1484.23dea70.76fb0000.dll
0x821dea70 explorer.exe        0x77bd0000 midimap.dll          OK: module.1484.23dea70.77bd0000.dll
0x821dea70 explorer.exe        0x7e1e0000 urlmon.dll           OK: module.1484.23dea70.7e1e0000.dll
0x821dea70 explorer.exe        0x74ad0000 POWRPROF.dll         OK: module.1484.23dea70.74ad0000.dll
0x821dea70 explorer.exe        0x7e410000 USER32.dll           OK: module.1484.23dea70.7e410000.dll
0x821dea70 explorer.exe        0x7e290000 SHDOCVW.dll          OK: module.1484.23dea70.7e290000.dll
0x821dea70 explorer.exe        0x77a20000 cscui.dll            OK: module.1484.23dea70.77a20000.dll
0x821dea70 explorer.exe        0x71aa0000 WS2HELP.dll          OK: module.1484.23dea70.71aa0000.dll
0x821dea70 explorer.exe        0x71cd0000 NETUI0.dll           OK: module.1484.23dea70.71cd0000.dll
0x821dea70 explorer.exe        0x71b20000 MPR.dll              OK: module.1484.23dea70.71b20000.dll
0x821dea70 explorer.exe        0x76360000 WINSTA.dll           OK: module.1484.23dea70.76360000.dll
```

Figure 11: Dlldump is used to extract dll information files for Kali.

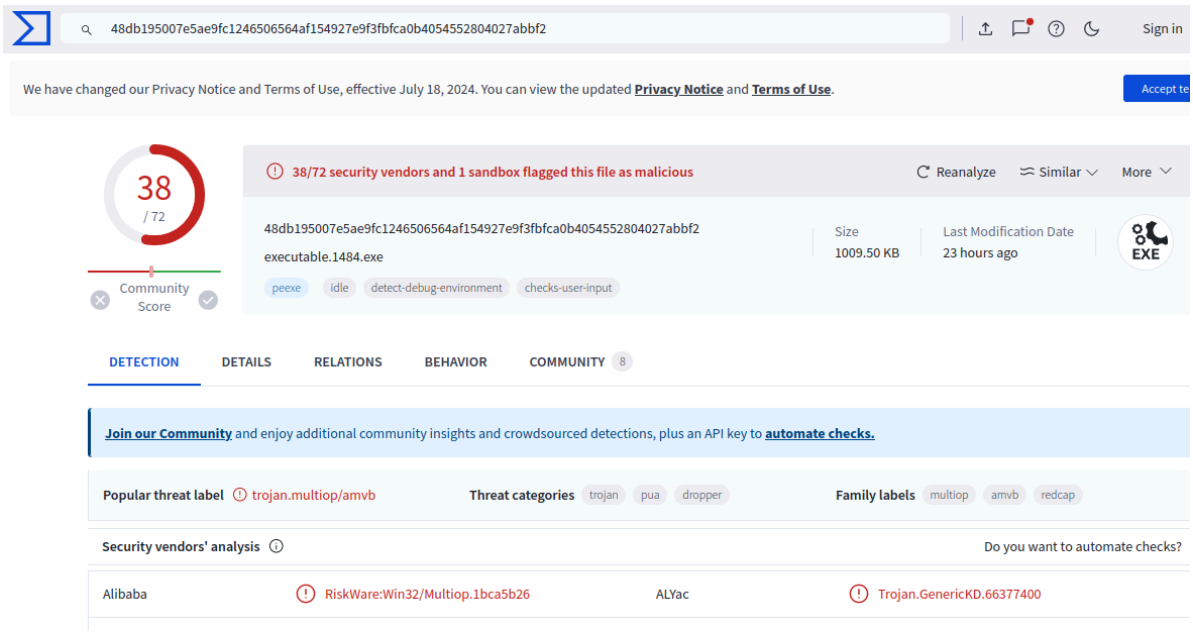


Figure 12: executable file examined in [3].

4. Extract the executables for one of the suspicious processes.

As we have already studied explorer.exe in-depth, we will continue this process for question 4. To extract executables for this suspicious process, we must dump the process's memory into a format we can manipulate to isolate and extract executables. As shown in Figure 13, using Volatility's memdump functionality to create a dmp file and then utilizing foremost to extract the files within it, we can then isolate the executables stored in the exe folder designated.

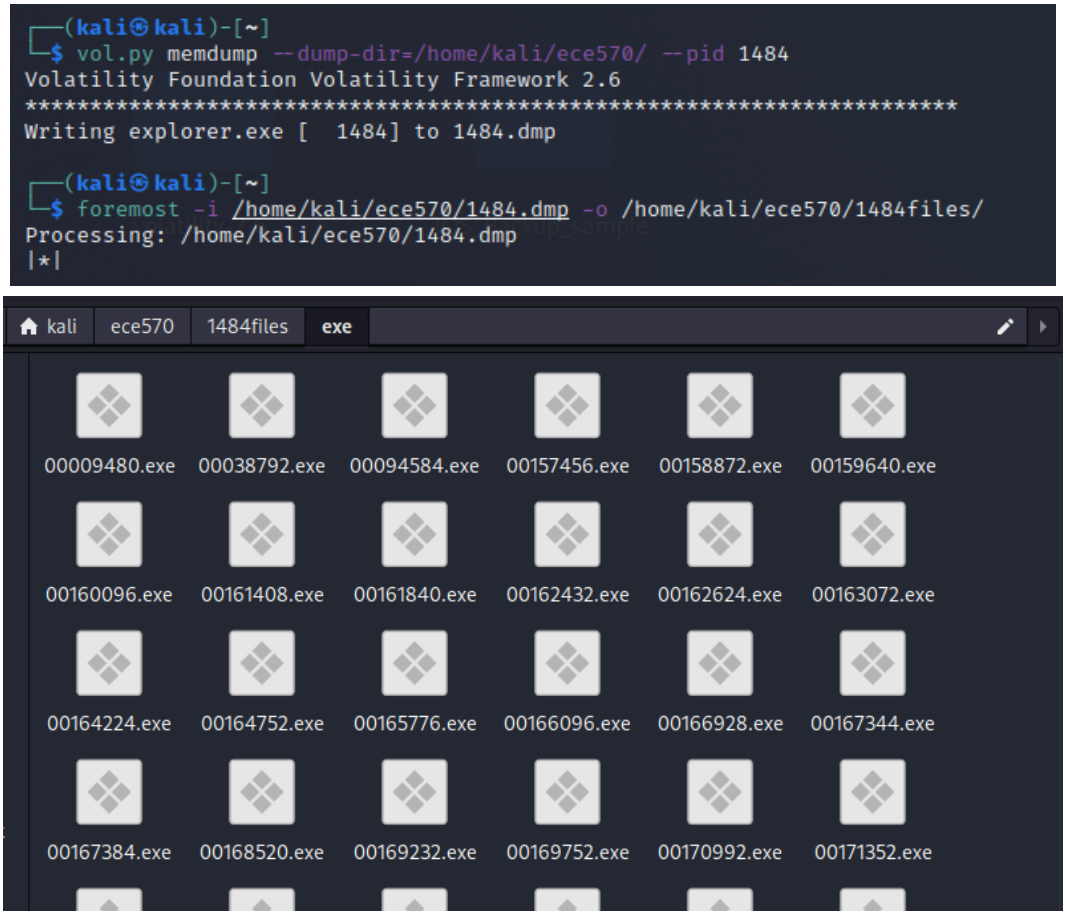


Figure 13: memdump and foremost utilization to extract Executables from explorer.exe

Through trial and error, we ran the process executables through [3], and we gathered that certain executables have flags indicating that some security vendors report them as possibly malicious. The greatest offender of this is executable file 00009480.exe, which, as shown in Figure 14, flags 31/72 security vendors as malicious.

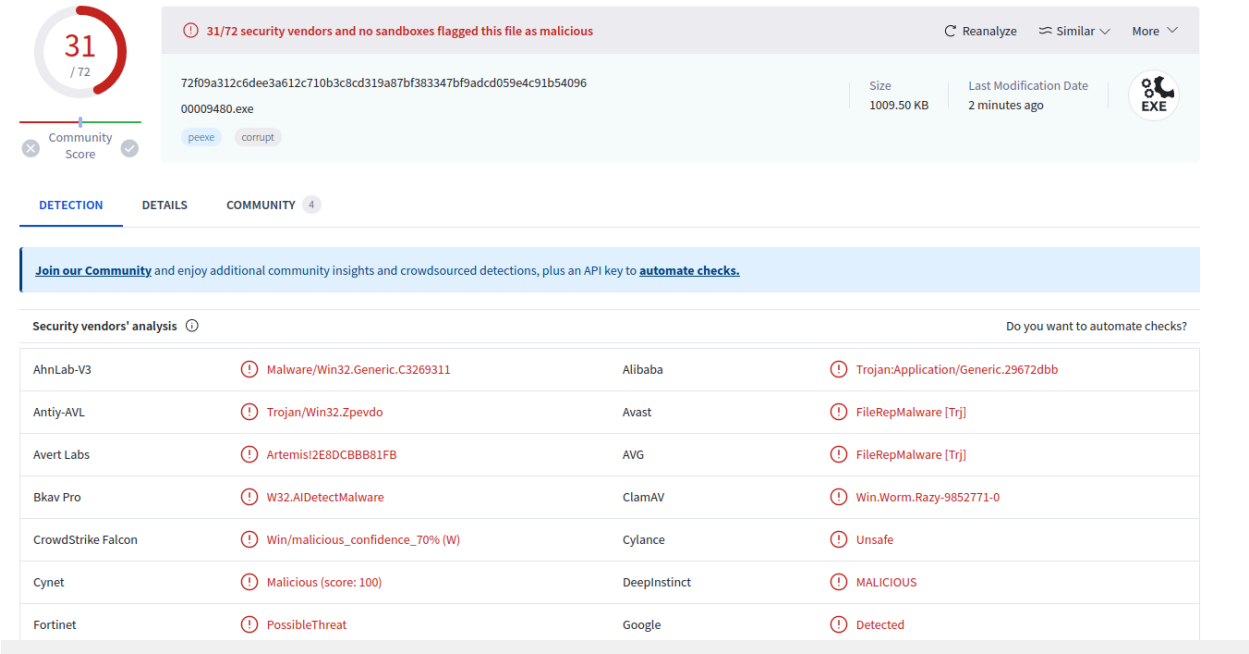


Figure 14: 00009480.exe anti-virus scan analysis in [3].

5. Identify the URLs and IPs for possible remote C&C servers.

To examine the URLs of possible command and control servers, we must examine the memory dump file of the suspicious processes. As we have already dumped the memory of the explorer.exe process, we will limit our search to this process as it is related to the initial winlogon.exe exploit. To examine the URLs within the dmp file, we will run the "strings 1484.dmp | grep http://" command, as shown in Figure 15 below.

```
┌──(kali㉿kali)-[~]
└─$ strings /home/kali/ece570/1484.dmp | grep "http://"
$http://www.trustcenter.de/guidelines0
'http://www.certplus.com/CRL/class3P.crl0
$http://crl.verisign.com/pca1.1.1.crl0G
http://www.usertrust.com1
http://www.usertrust.com1
3http://crl.usertrust.com/UTN-USERFirst-Hardware.crl01
http://www.valicert.com/1 0
http://www.valicert.com/1 0
(http://www.certplus.com/CRL/class3TS.crl0
*http://ca.sia.it/seccli/repository/CRL.der0J
http://www.usertrust.com1
http://www.usertrust.com1
,http://crl.usertrust.com/UTN-DATACorpSGC.crl0*
http://www.usertrust.com1+0)
```

Figure 15: grep http:// for explorer.exe memory dump.

As we know, this generates a long list of URLs embedded within the process, and we need to siphon through them to determine possible C&C server URLs. While completing this process, one URL appeared in numerous instances within the code http://188.40.0.138:8080/zb/v_01_a/in/cp.php, as shown in Figure 16.

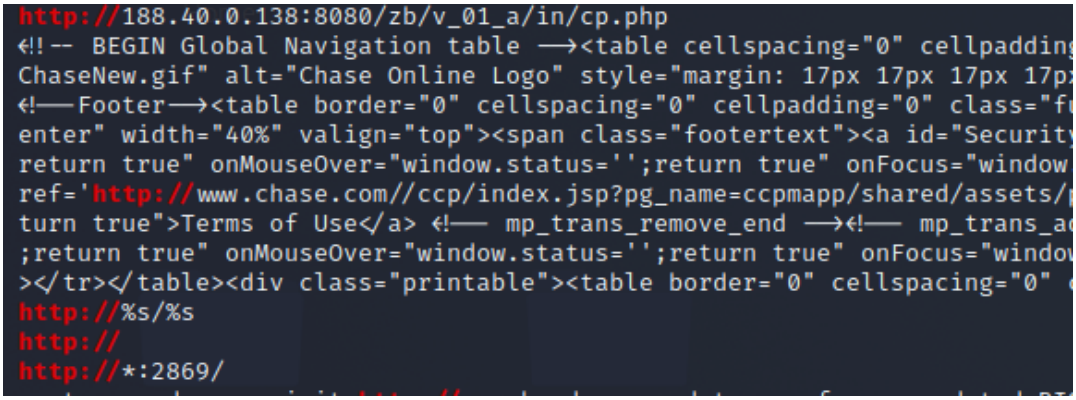


Figure 16: IP embedded within memdump file for explorer.exe

To take things one step further, we conducted a string search on ":8080" to see if any other IP addresses appeared in the memdump file. As we see in Figure 17, two additional IP addresses flag this: 41.168.5.140 and 125.19.103.198. Strangely enough, they have the same file path as our URL, implying some link between these IPs.

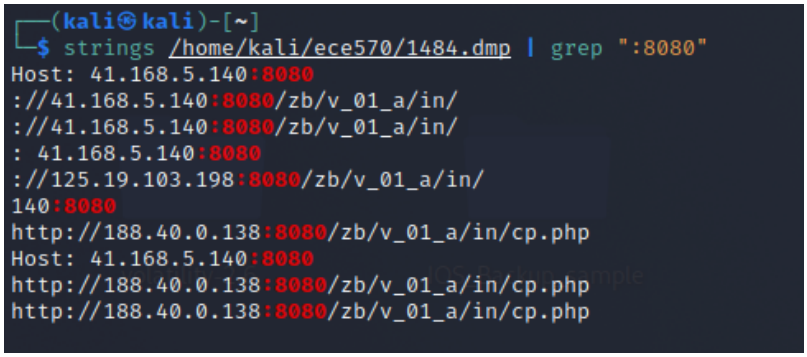


Figure 17: string search for IPs in explorer.exe process.

Returning to volatility, we further wanted to see if there were active connections to sockets which can indicate the originating C&C server. In Figure 18, we utilized connections and connscan functionality of volatility and found connections between two of the aforementioned IPs, 41.168.5.140:8080 and 125.19.103.198:8080, through the explorer.exe process.

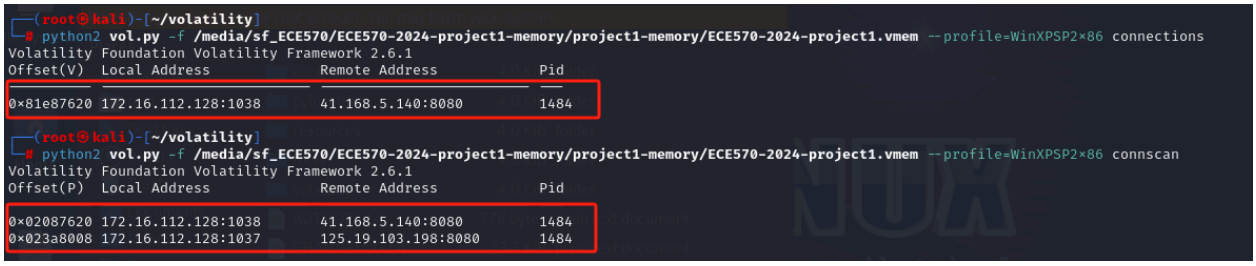


Figure 18: connections and connscan of memory dump file.

Therefore, we now have three possible IPs to examine as possible C&C servers, and we proceed to investigate these using [3] to gain additional insight into the IPs. As noted in Figure 19, 188.40.0.138 only flags 1 security vendor as malicious and originates from Germany, 125.19.103.198 flags 3 and originates from India, and 41.168.5.140 flags 4 and originates from South Africa. This confirms that the servers at these addresses are possible C&C servers.

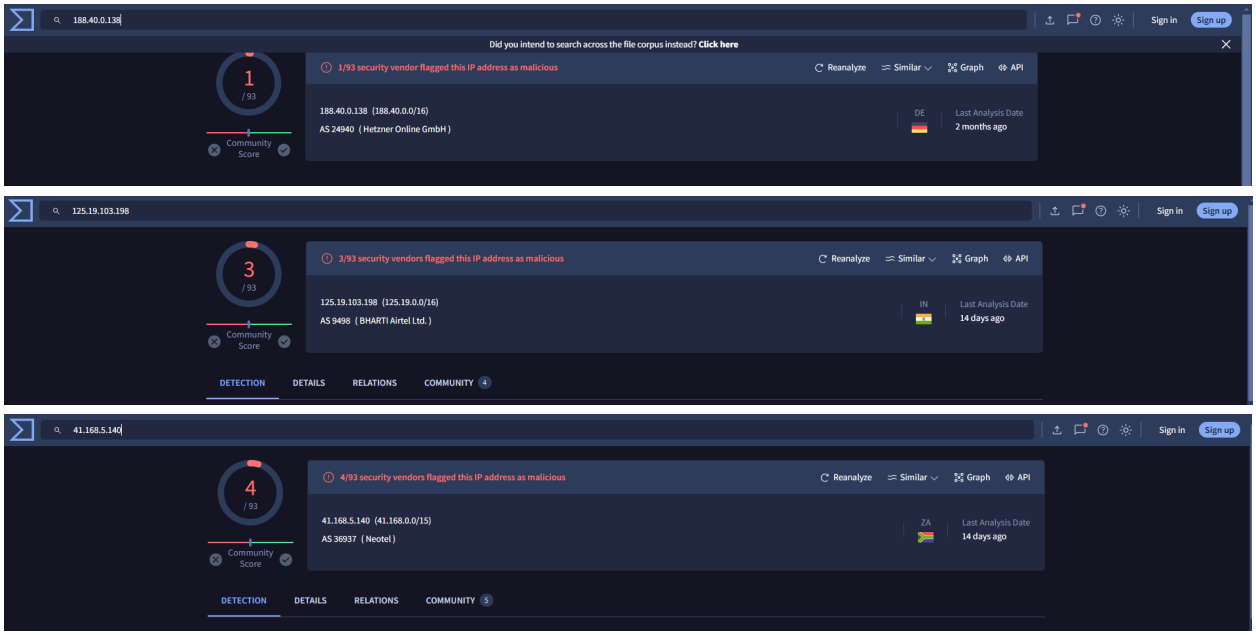


Figure 19: IP examination of possible C&C servers in the explorer.exe process.

6. Identify a potentially malicious hive from the list.

Using the hivelist functionality of Volatility, we can list all available hives, as shown in Figure 20.

```
┌──(kali㉿kali)-[~]
└─$ vol.py hivelist
Volatility Foundation Volatility Framework 2.6
Virtual    Physical   Name
---------- ---------- ----

0×e18e5b60 0×093f8b60 \Device\HarddiskVolume1\Documents and Settings\Robert\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
0×e1a19b60 0×0a5a9b60 \Device\HarddiskVolume1\Documents and Settings\Robert\NTUSER.DAT
0×e18398d0 0×08a838d0 \Device\HarddiskVolume1\Documents and Settings\LocalService\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
0×e18614d0 0×08e624d0 \Device\HarddiskVolume1\Documents and Settings\LocalService\NTUSER.DAT
0×e183bb60 0×08e2db60 \Device\HarddiskVolume1\Documents and Settings\NetworkService\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
0×e17f2b60 0×08519b60 \Device\HarddiskVolume1\Documents and Settings\NetworkService\NTUSER.DAT
0×e1570510 0×07669510 \Device\HarddiskVolume1\WINDOWS\system32\config\software
0×e1571008 0×0777f008 \Device\HarddiskVolume1\WINDOWS\system32\config\default
0×e15709b8 0×076699b8 \Device\HarddiskVolume1\WINDOWS\system32\config\SECURITY
0×e15719e8 0×0777f9e8 \Device\HarddiskVolume1\WINDOWS\system32\config\SAM
0×e13ba008 0×02e4b008 [no name]
0×e1035b60 0×02ac3b60 \Device\HarddiskVolume1\WINDOWS\system32\config\system
0×e102e008 0×02a7d008 [no name]
```

Figure 20: Available registry hives for memdump.

As we know from previous questions, our initial exploit, winlogon.exe, launched on 2012-07-22 at 02:42:32. Based on the “Last updated” parameter shown in the print key Volatility function, we can infer which hives could be malicious and match that with our winlogon.exe launch time. Using the specific date as a restricting factor, two possible registry hives could be malicious, as shown in Figure 21.

```
Values:
-------------------------
Registry: \Device\HarddiskVolume1\Documents and Settings\Robert\NTUSER.DAT
Key name: $$$PROTO.HIV (S)
Last updated: 2012-07-22 02:42:37 UTC+0000

Subkeys:
  (S) AppEvents
  (S) Console
  (S) Control Panel
  (S) Environment
  (S) Identities
  (S) Keyboard Layout
  (S) Printers
  (S) Software
  (S) UNICODE Program Groups
  (S) Windows 3.1 Migration Status
  (V) SessionInformation
  (V) Volatile Environment
```

```
Values:
-------------------------
Registry: \Device\HarddiskVolume1\WINDOWS\system32\config\SECURITY
Key name: SECURITY (S)
Last updated: 2012-07-22 02:42:32 UTC+0000

Subkeys:
  (S) Policy
  (S) RXACT
  (V) SAM
```

Figure 21: Hive Registries modified after winlogon.exe starts.

To examine these two registries more closely, we will employ RegRipper on registry dumps after running them with the registrydump command in Volatility, as demonstrated in Figure 22.

```
┌──(kali㉿kali)-[~]
└─$ vol.py dumpregistry -o 0×e15709b8 --dump-dir=/home/kali/ece570
Volatility Foundation Volatility Framework 2.6
**************************************************
Writing out registry: registry.0×e15709b8.SECURITY.reg

**************************************************
```

Figure 22: dumpregistry for the suspicious hive.

Examining the generated reports from RegRipper, as shown in Figure 23, alludes to some concerning information, specifically about the NTUSER.DAT Registry.

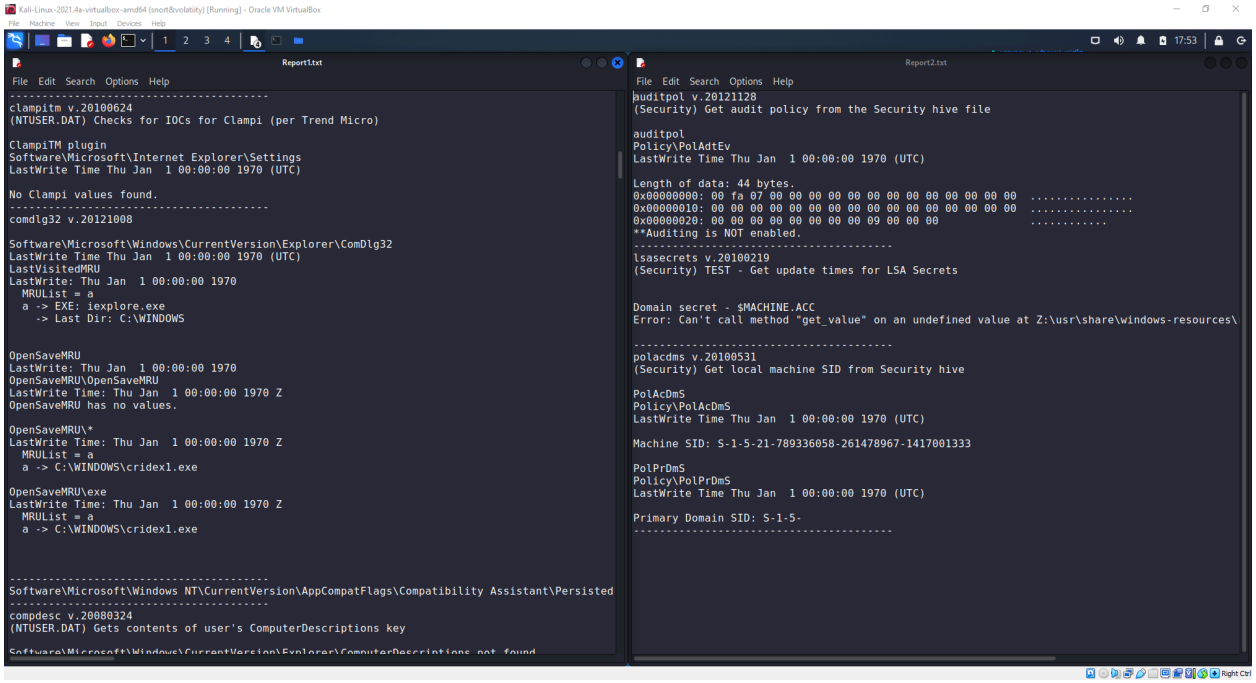


Figure 23: RegRipper Reports.

As shown in the Figure on the left side in report one, we highlight a region of the registry, which depicts the utilization of cridex1.exe in a Most Recently Used list for the application. As noted by Microsoft, Cridex is a known malware worm; thus, our suspicions of this registry are confirmed.[4] Focusing our attention more on the Report1.txt generated by the NTUSER.DAT Registry, if we looked more within the file at the UserAssist region designated by RegRipper, we could see cridex1.exe as a run path executable as shown in Figure 24, further perpetuating that it was recently run on the system resulting in the propagation of the worm.

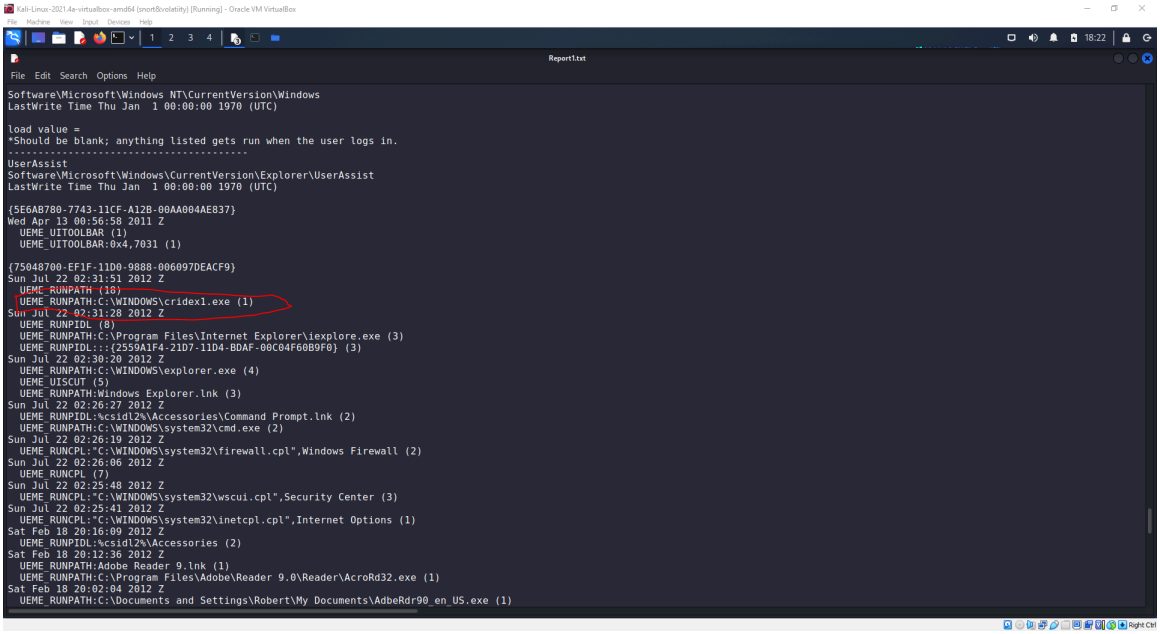


Figure 24: UserAssist region of report with cridex1.exe circled.

References

[1] "Core processes in Windows system," CyberY, https://0xcybery.github.io/blog/Core-Processes-In-Windows-System (accessed May 29, 2024).

[2] "Mz Exe Files," MZ EXE files · Cogs and Levers, https://tuttlem.github.io/2015/03/28/mz-exe-files.html (accessed May 29, 2024).

[3] "Virustotal," VirusTotal, https://virustotal.com/ (accessed May 29, 2024).

[4] Win32/CRIDEX threat description - microsoft security intelligence, https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32/Cridex (accessed May 29, 2024).