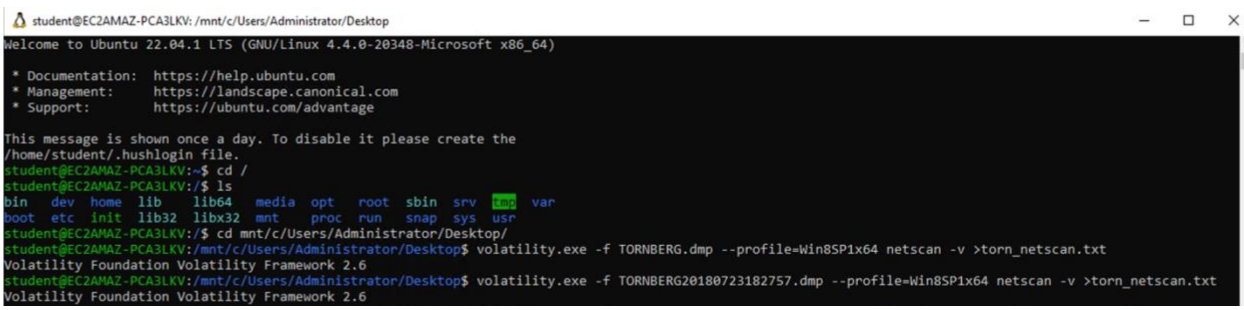
Volatility -f TORNBERG.dmp --profile=Win8SP1x64 netscan -v >torn_netscan.txt

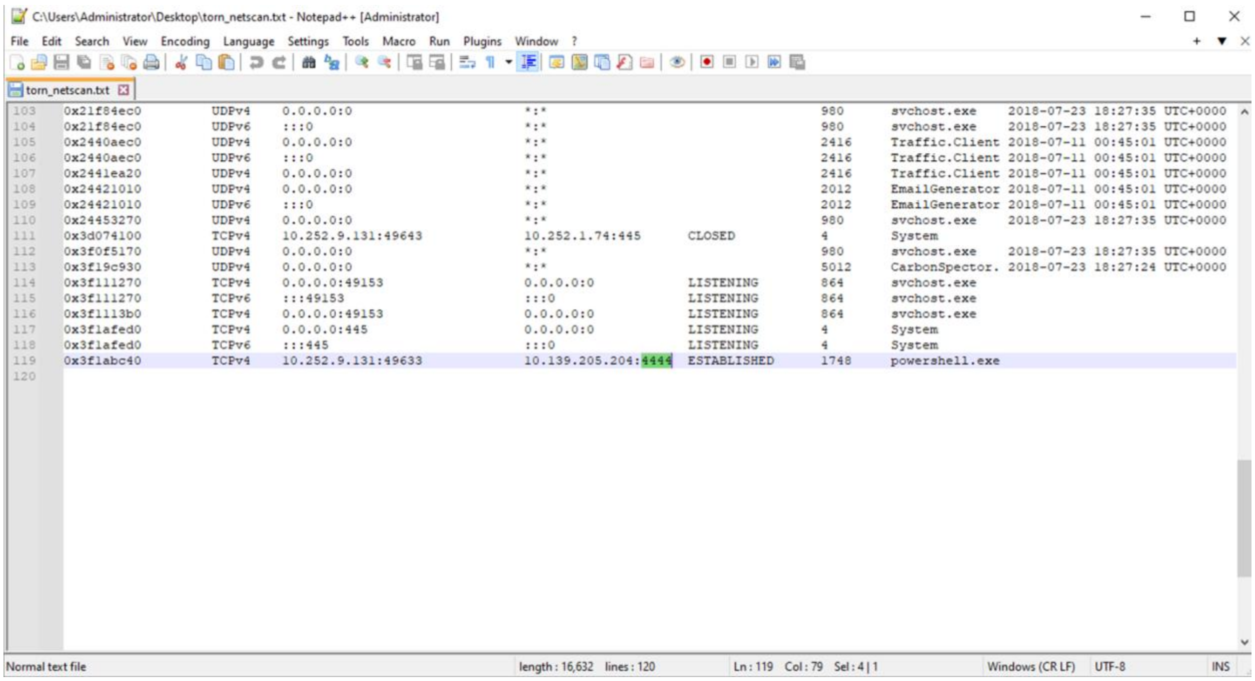
In this task, the objective was to analyze network connections from a memory dump using the volatility tool within the Kali Linux Windows Subsystem for Linux. The project involved extracting data from a memory file called TORNBERG.dmp to identify network connections and pinpoint the process responsible for an established connection on a specific remote port.

To carry out this process, the following command was used:

volatility -f TORNBERG.dmp --profile=Win8SP1x64 netscan -v >torn_netscan.txt



This command generates a text file torn_netscan.txt containing a detailed list of all active and listening network connections that existed at the time the memory capture was taken. By opening this file in notepad++, we were able to examine the network activity and focus on identifying a connection made to remote port 4444.

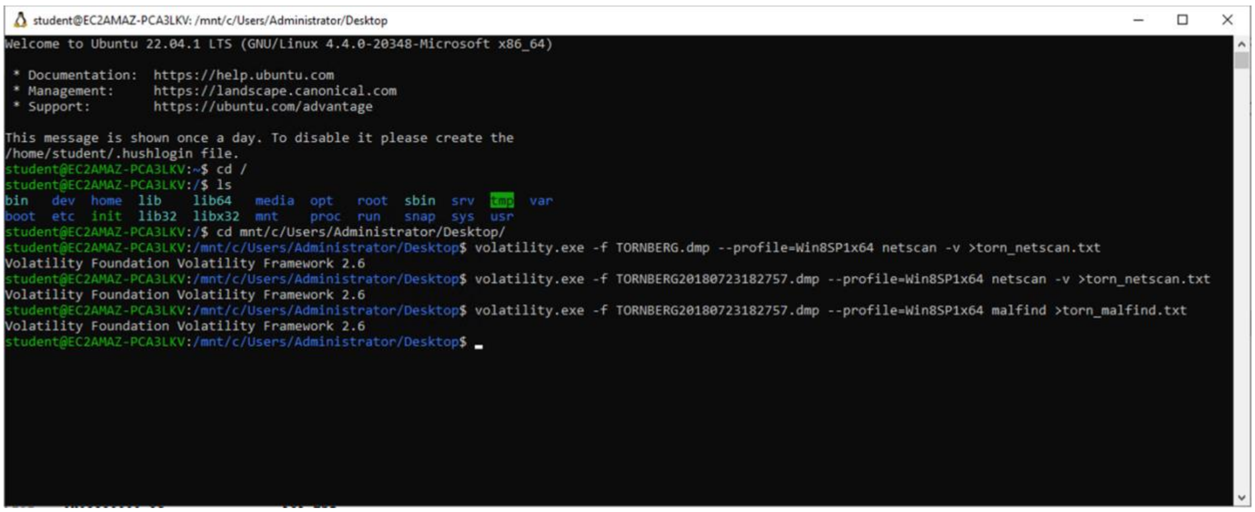


Upon reviewing the file, the relevant connection was found on line 119. This shows that the local IP address 10.252.9.131 on port 49633 was connected to the remote IP address 10.139.205.204 on port 4444. The connection is marked as established, and it is linked to the process powershell.exe.

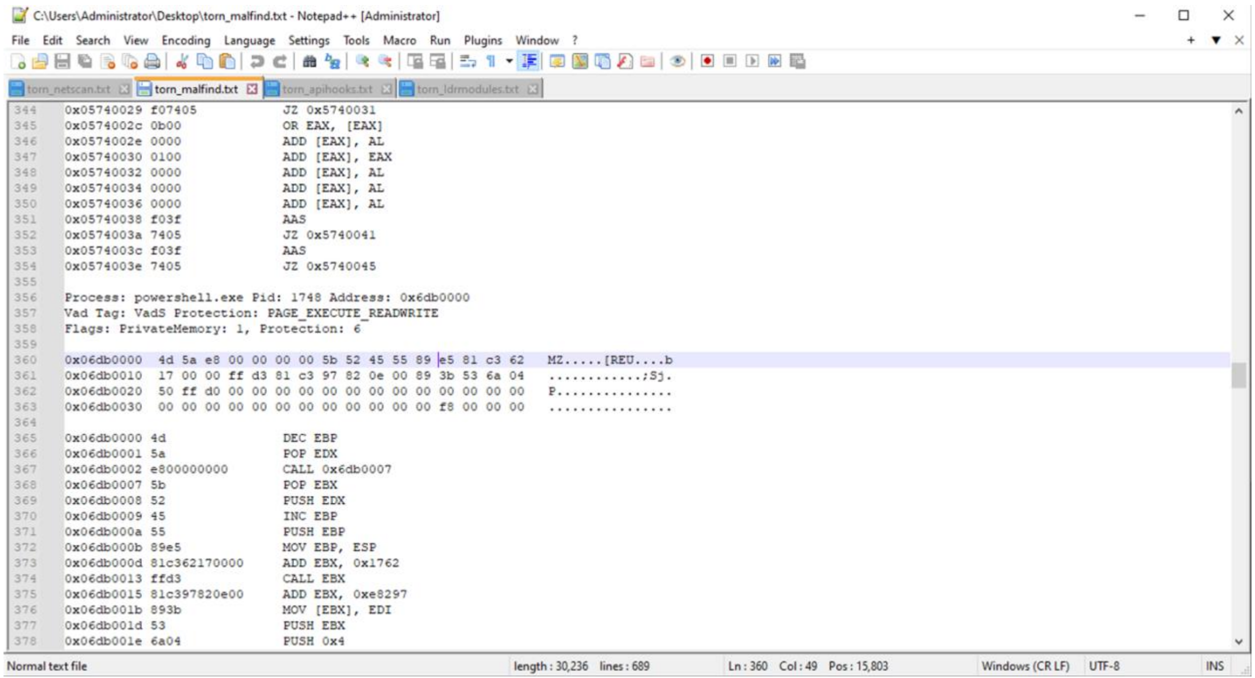
The Process ID or PID for this particular connection is **1748**. This identifier is key to further analysis, allowing for tracking of the specific process that was responsible for the network activity. Powershell.exe is often used in system operations but can also be leveraged for nefarious purposes, making it an important detail in any network or system investigation.

Volatility in the Kali Linux Windows Subsystem for Linux

In this task, the objective is to investigate a potential case of process injection using a memory dump file, with the help of the volatility toolset in a Kali Linux environment. Specifically, we need to identify the process responsible for injecting malicious code into a powershell.exe process with PID 1748. The correct answer, which emerges after carefully analyzing the memory artifacts, is 5012. The steps involved in reaching this conclusion are detailed below.



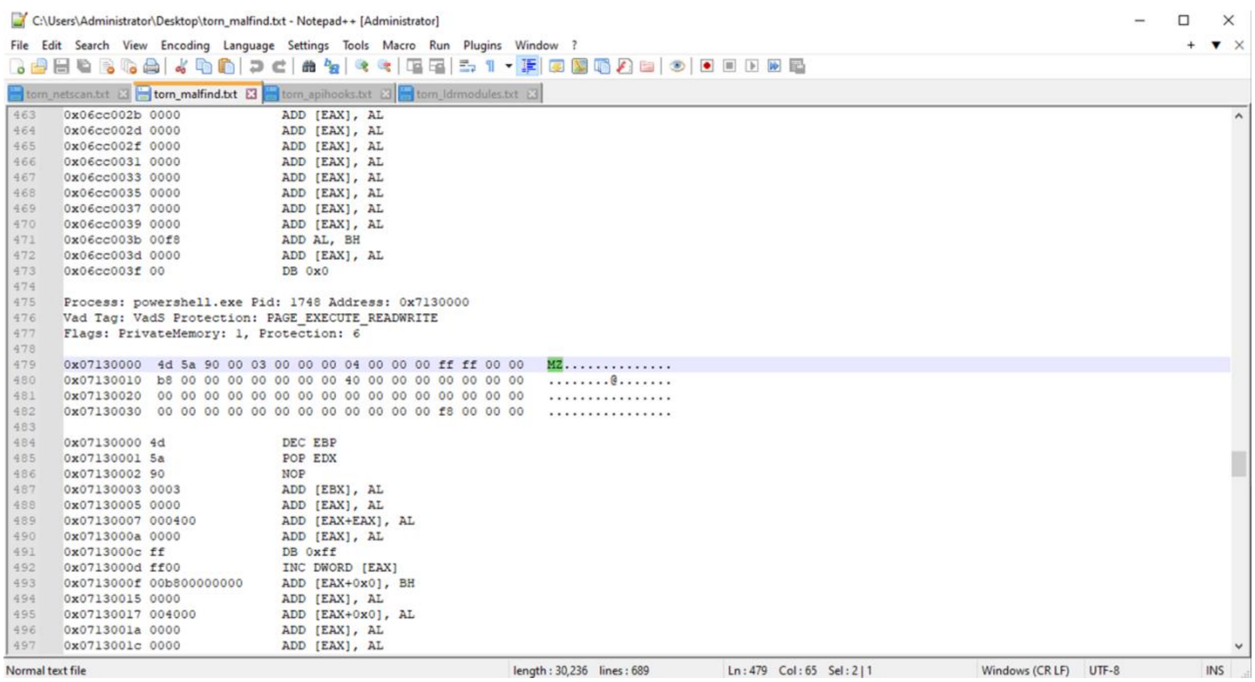
The first step in the investigation is to run the malfind plugin on the memory dump. This command allows us to detect processes that show signs of being manipulated or injected with malicious code. After searching the output for entries related to PID 1748, we locate two relevant memory addresses. One of these addresses stands out because the data begins with the "MZ" signature, which indicates a Windows executable header. This signature is important because it suggests that the memory contains executable code, possibly the result of process injection.



C:\Users\Administrator\Desktop\torn_malfind.txt - Notepad++ [Administrator]

```
0x05740029 f07405           JZ 0x5740031
0x0574002c 0b00             OR EAX, [EAX]
0x0574002e 0000             ADD [EAX], AL
0x05740030 0100             ADD [EAX], EAX
0x05740032 0000             ADD [EAX], AL
0x05740034 0000             ADD [EAX], AL
0x05740036 0000             ADD [EAX], AL
0x05740038 f03f             AAS
0x0574003a 7405             JZ 0x5740041
0x0574003c f03f             AAS
0x0574003e 7405             JZ 0x5740045

Process: powershell.exe Pid: 1748 Address: 0x6db0000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: PrivateMemory: 1, Protection: 6

0x06db0000  4d 5a e8 00 00 00 00 5b 52 45 55 89 e5 81 c3 62   MZ.....[REU....b
0x06db0010  17 00 00 ff d3 81 c3 97 82 0e 00 89 3b 53 6a 04   ............;Sj.
0x06db0020  50 ff d0 00 00 00 00 00 00 00 00 00 00 00 00 00   P...............
0x06db0030  00 00 00 00 00 00 00 00 00 00 00 00 f8 00 00 00   ................

0x06db0000 4d               DEC EBP
0x06db0001 5a               POP EDX
0x06db0002 e800000000       CALL 0x6db0007
0x06db0007 5b               POP EBX
0x06db0008 52               PUSH EDX
0x06db0009 45               INC EBP
0x06db000a 55               PUSH EBP
0x06db000b 89e5             MOV EBP, ESP
0x06db000d 81c362170000     ADD EBX, 0x1762
0x06db0013 ffd3             CALL EBX
0x06db0015 81c397820e00     ADD EBX, 0xe8297
0x06db001b 893b             MOV [EBX], EDI
0x06db001d 53               PUSH EBX
0x06db001e 6a04             PUSH 0x4
```

Normal text file length : 30,236 lines : 689 Ln : 360 Col : 49 Pos : 15,803 Windows (CR LF) UTF-8 INS

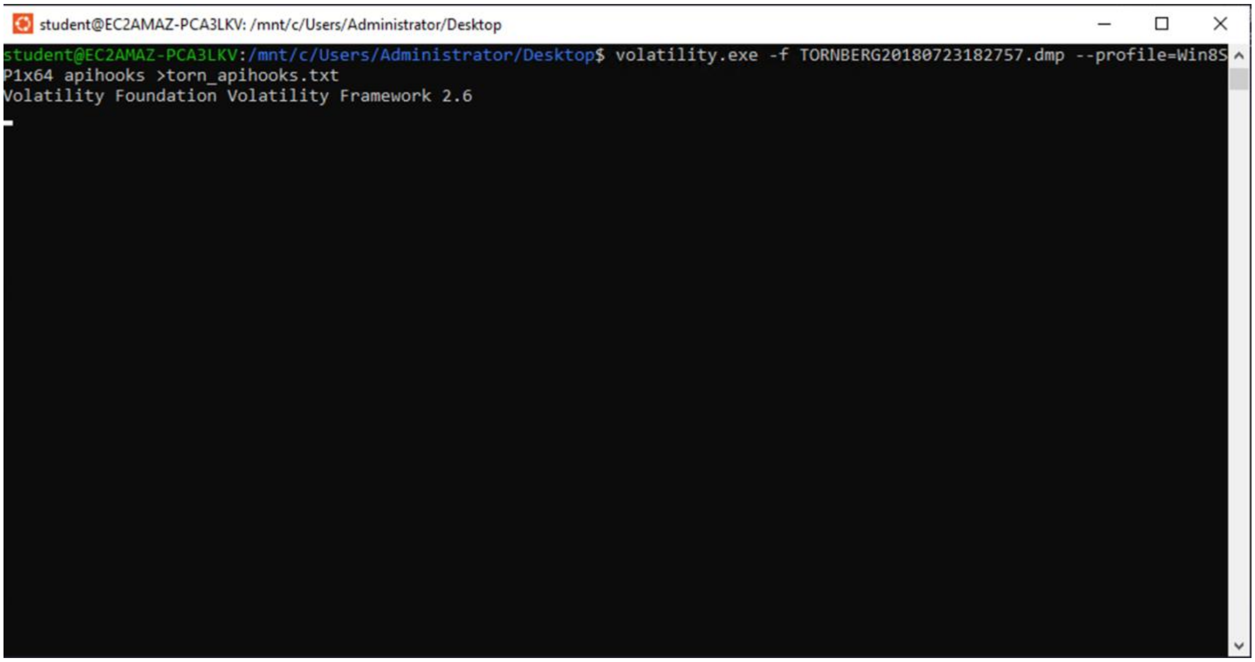


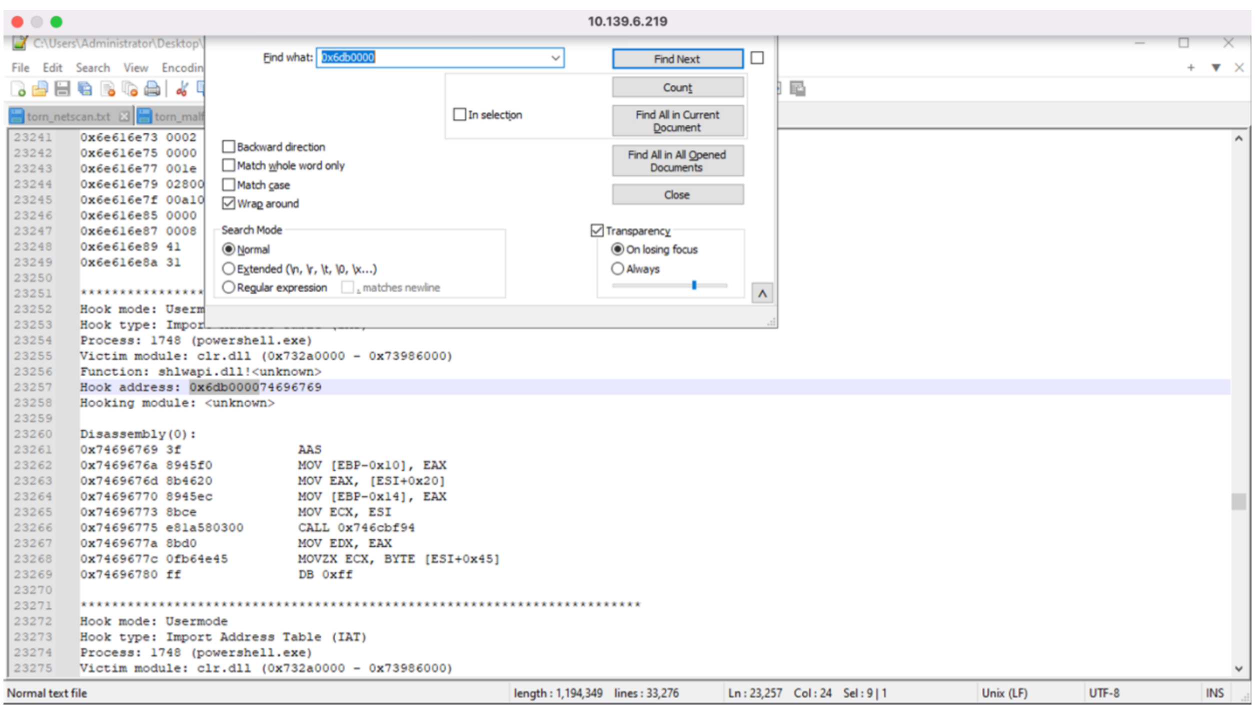
C:\Users\Administrator\Desktop\torn_malfind.txt - Notepad++ [Administrator]

```
0x06cc002b 0000             ADD [EAX], AL
0x06cc002d 0000             ADD [EAX], AL
0x06cc002f 0000             ADD [EAX], AL
0x06cc0031 0000             ADD [EAX], AL
0x06cc0033 0000             ADD [EAX], AL
0x06cc0035 0000             ADD [EAX], AL
0x06cc0037 0000             ADD [EAX], AL
0x06cc0039 0000             ADD [EAX], AL
0x06cc003b 00f8             ADD AL, BH
0x06cc003d 0000             ADD [EAX], AL
0x06cc003f 00               DB 0x0

Process: powershell.exe Pid: 1748 Address: 0x7130000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: PrivateMemory: 1, Protection: 6

0x07130000  4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00   MZ..............
0x07130010  b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00   ........@.......
0x07130020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x07130030  00 00 00 00 00 00 00 00 00 00 00 00 f8 00 00 00   ................

0x07130000 4d               DEC EBP
0x07130001 5a               POP EDX
0x07130002 90               NOP
0x07130003 0003             ADD [EBX], AL
0x07130005 0000             ADD [EAX], AL
0x07130007 000400           ADD [EAX+EAX], AL
0x0713000a 0000             ADD [EAX], AL
0x0713000c ff               DB 0xff
0x0713000d ff00             INC DWORD [EAX]
0x0713000f 00b800000000     ADD [EAX+0x0], BH
0x07130015 0000             ADD [EAX], AL
0x07130017 004000           ADD [EAX+0x0], AL
0x0713001a 0000             ADD [EAX], AL
0x0713001c 0000             ADD [EAX], AL
```

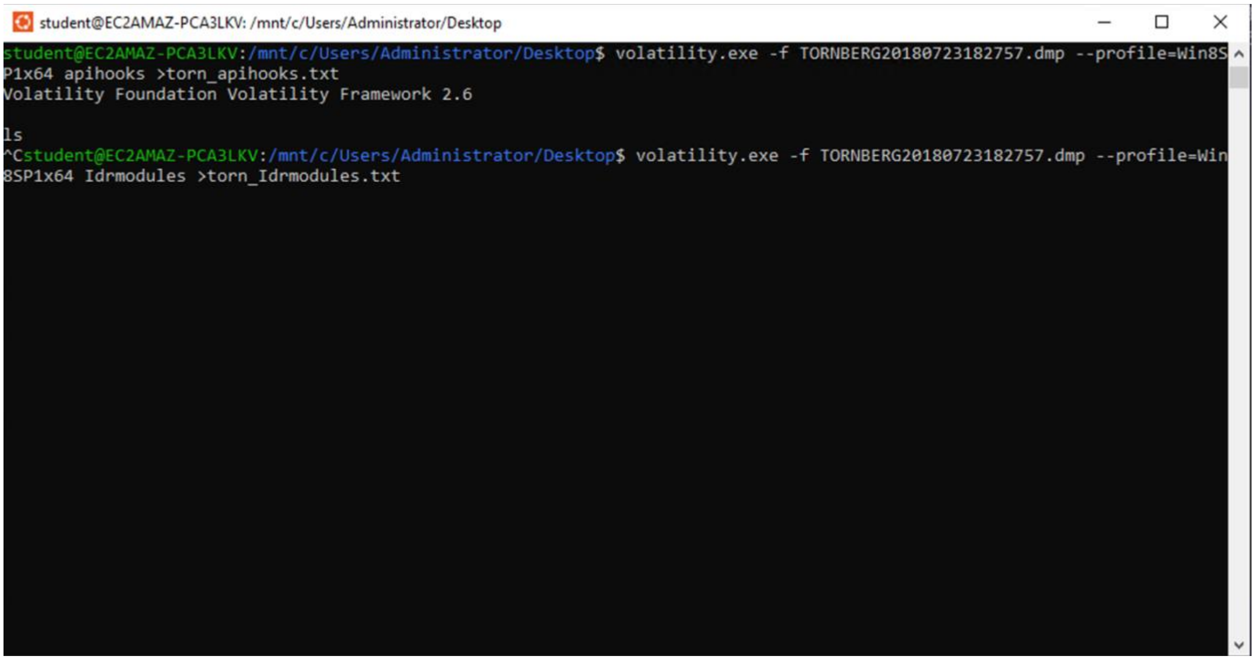
Normal text file length : 30,236 lines : 689 Ln : 479 Col : 65 Sel : 2 | 1 Windows (CR LF) UTF-8 INS

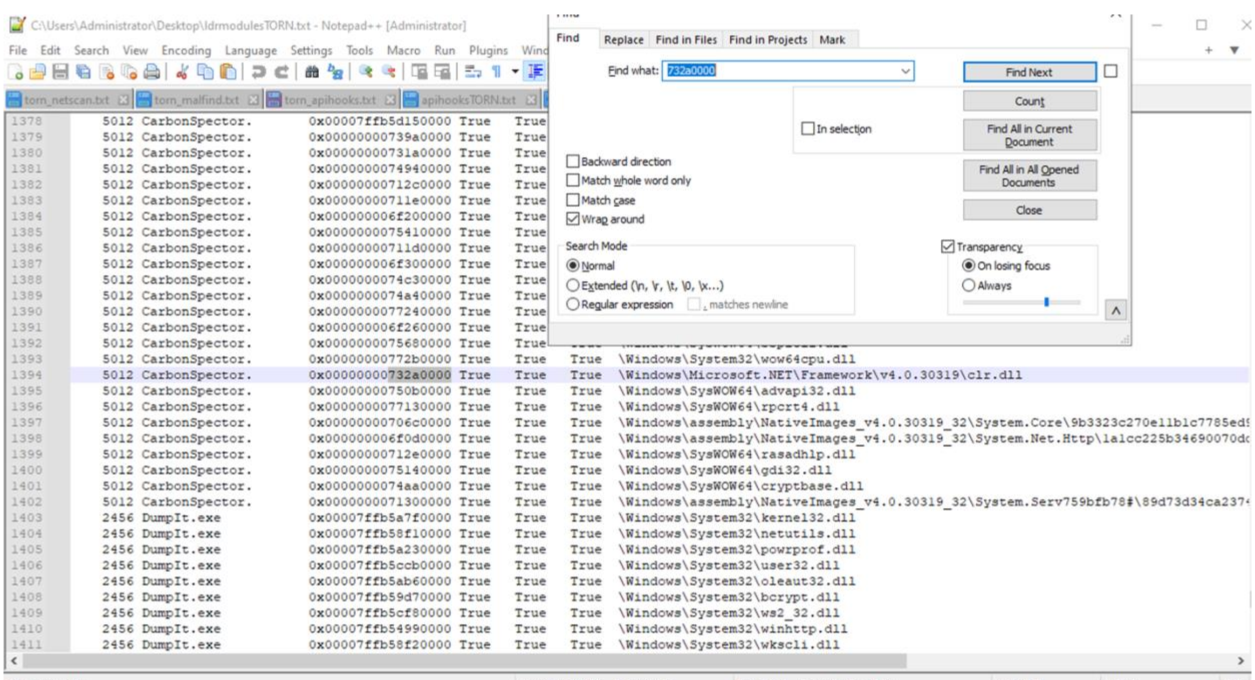
Next, we analyze this memory region further by using the apihooks plugin. This plugin scans for API hooking activity, a technique commonly used in malicious operations to intercept system function calls.





In this case, one of the memory addresses from the malfind output is flagged in the apihooks results, confirming that API hooking has occurred at this location. This is a critical finding, as it points to the likelihood that malicious activity is taking place within the hooked memory region. The apihooks output further reveals that the hooked address is associated with a DLL located at 0x732a0000.





The next step involves determining which processes have loaded this suspicious DLL. To do this, we run the ldrmodules plugin, which lists all modules (such as DLLs) loaded into the memory of each process. By searching for 0x732a0000, we find that this DLL is present in two processes: PID 1748, the target process powershell.exe, and another process PID 5012. This duplication strongly suggests that PID **5012** played a role in injecting the DLL into PID 1748.

Netscan Module on the Tornberg File.

In this task, we examined the memory image using the Volatility netscan module to analyze the network activity that took place at the time the memory was captured. The goal was to interpret the data and determine which statements accurately reflect the information found within the network scan results. Three specific statements have been identified as correct, based on the evidence provided by the analysis.

The first correct observation is that the IP address of the host system is 10.252.9.131. By reviewing the output of the netscan module, we can see this IP address associated with the local machine in several network connections. These connections indicate that the host's internal IP address within the network environment at the time of the capture was 10.252.9.131. This IP consistently appears as the local address in various communication sessions, solidifying it as the IP of the host system.

The second accurate statement relates to the process responsible for handling the remote connection, which is svchost.exe. In Windows environments, svchost.exe is a critical process that manages many essential system services. In this case, it was responsible for managing the network service tied to the remote connection. The netscan results reveal that svchost.exe was linked to a network connection commonly associated with Remote Desktop Protocol (RDP), confirming that it handled the session. This is a typical role for svchost.exe, which often manages remote service protocols such as RDP.

Lastly, the analysis confirms that the host machine received a remote desktop connection. This conclusion is drawn from the presence of an established connection on port 3389, which is the standard port used by RDP. The netscan output clearly shows that the host system was listening on this port and that a remote session was initiated, indicating that the host was acting as the RDP server. This connection implies that an external machine accessed the host system via RDP, confirming the statement that the host received a remote desktop connection.

Conclusion

In summary, we used the Volatility tool to examine a memory dump and investigate network and memory activity. The primary goal was to identify active network connections, processes involved, and detect possible cases of process injection. The netscan module revealed that the host system's IP address was 10.252.9.131, and the process svchost.exe was responsible for managing a remote desktop connection. An established connection on port 3389 confirmed that the host had received a remote desktop session. Additionally, using the malfind and apihooks plugins, we identified that the process powershell.exe (PID 1748) had been injected with suspicious code, as seen by the presence of an executable code signature. Further investigation through the ldrmodules plugin showed that another process, PID 5012, had loaded the same suspicious DLL, suggesting that PID 5012 was responsible for the injection. By combining these findings, we were able to trace key network activities and uncover the process injection, providing a detailed understanding of the system's state during the memory capture.