DLL Injection & IAT Hooking - Daniel Ayzenshteyn

Exercise 1: DLL Injection

Implementated the Injector.exe which uses DLL Injection for injection of arbitrary code into a target process and Injected.dll which changes the title in the current process.

The syntax is DLLInjector.exe <PID> <DLL_PATH>
Both arguments optional -

1. If PID not specified it Injects to all notepad.exe or mspaint.exe processes.
2. If DLL_PATH not specified in command line arguments then it uses the hard coded dll path in the code.

*Provided a video demonstration named “Notepad&PaintHijacking.mp4”.

** The reason for using paint is that Notepad on Windows 11 does not show the Title Bar as in Windows 10, However mspaint.exe still does show the title bar.
In the video demonstration, I demonstrated both applications. However, it should be noted that Notepad shows the title only in the Taskbar at the bottom of the screen.

Exercise 2: IAT Hooking

First of all I implemented IAT Patching to grasp the PE structure and get the idea of how it should work.

Then I transferred the IAT Patching code from a Console App to a DLL, so it will execute in the virtual space of the Injected process, thus giving us IAT Hooking.

I used the Injector from Exercise 1 to Inject my new IATHooking.dl. Then I tested it on Notepad.exe to verify that I get code execution in the address space of the injected process (Notepad.exe) - I used the find functionality (CTRL+F), which pops up a window after you click “Find All”. My hooked message box popped up, confirming that my IAT hooking work.

For the creativity part I chose to hide TCP connections from tools like NETSTAT.EXE.
*Provided a video demonstration named “CMD&NetstatHijacking.mp4”.

summary of creative part

I hooked CreateProcessW in cmd.exe via DLL Injection. The hooked CreateProcessW will create the process in SUSPENDED mode, Inject a DLL to the child process and then resume it’s run. (Similar to Process Hollowing 🙂).
Then the Injected DLL in the child process (Netstat.exe) will try to retrieve the connections and those functions will be hooked because we hooked them while it was in suspended mode, and before the child process had time to run anything.
I also implemented a HideInject.exe which unloads the DLL from the Injected cmd.exe thus, returning it to its normal state (This could be done via saving the original function addresses before hooking them, and then in process DETACH we hook the IAT back to its original functions. HideInject.exe is implemented in such a way that it get’s the DLL address from a

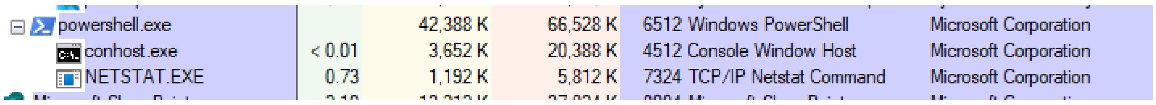
remote process and then calls FreeLibrary with a remote thread - thus HideInject.exe only needs to detach the malicious DLL from the target process for stealth).

Appendix

Here I provide you some of the steps and challenges (some of them yet to be solved) I faced during the exercise.

I.

First I tried to spot how does powershell.exe and cmd.exe use netstat.exe, and I noticed some weird behavior… sometimes a new process is created and sometimes it didn't - After research it concludes to a point that powershell implementation of netstat is inner so it doesn't always uses NETSTAT.exe binary… and worse than that it doesn't even use CreateProcess so it is unclear what functionality needs to be hooked to achieve Injection in child process.

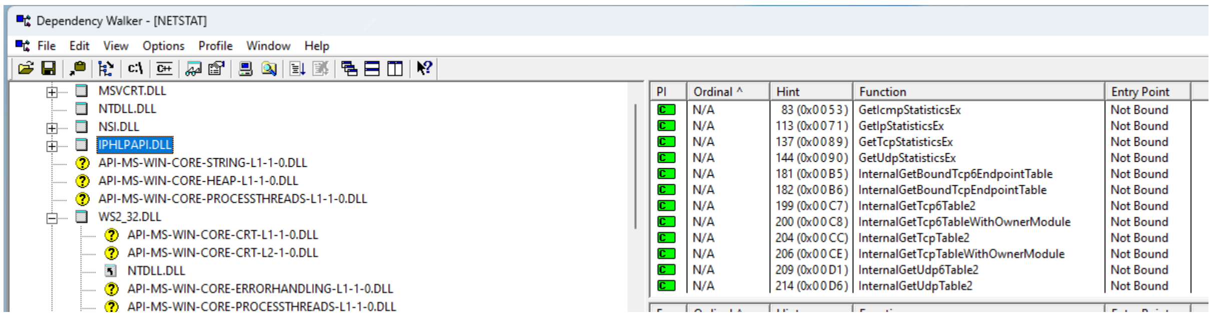


```
powershell.exe              42,388 K   66,528 K   6512 Windows PowerShell          Microsoft Corporation
  conhost.exe      < 0.01    3,652 K   20,388 K   4512 Console Window Host         Microsoft Corporation
  NETSTAT.EXE        0.73    1,192 K    5,812 K   7324 TCP/IP Netstat Command      Microsoft Corporation
```

I found that cmd.exe DOES use CreateProcessW so we will take it from here to cmd.exe and hook it's CreateProcessW so we will be able to control the functionality of NETSTAT.exe and other similar applications.

II.

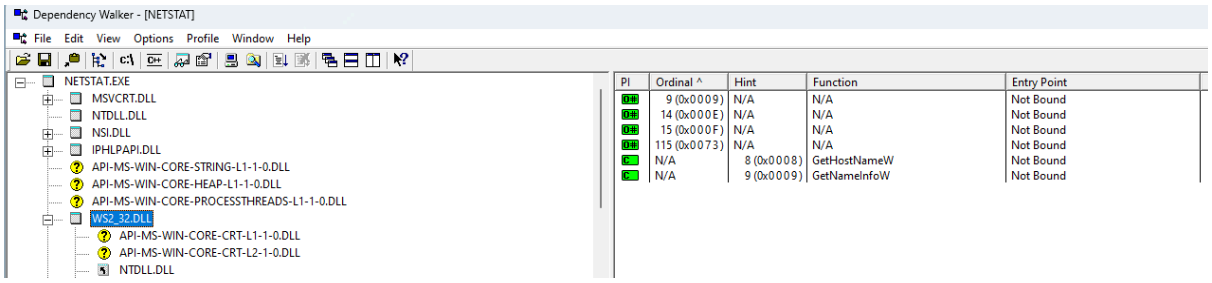
The next step was to track down what API calls NETSTAT.exe makes so we can hook those functions and hopefully change their outputs (to hide malicious connections).



Dependency Walker - [NETSTAT]

```
MSVCRT.DLL
NTDLL.DLL
NSI.DLL
IPHLPAPI.DLL
API-MS-WIN-CORE-STRING-L1-1-0.DLL
API-MS-WIN-CORE-HEAP-L1-1-0.DLL
API-MS-WIN-CORE-PROCESSTHREADS-L1-1-0.DLL
WS2_32.DLL
    API-MS-WIN-CORE-CRT-L1-1-0.DLL
    API-MS-WIN-CORE-CRT-L2-1-0.DLL
    NTDLL.DLL
    API-MS-WIN-CORE-ERRORHANDLING-L1-1-0.DLL
    API-MS-WIN-CORE-PROCESSTHREADS-L1-1-0.DLL
```

PI	Ordinal ^	Hint	Function	Entry Point
C	N/A	83 (0x0053)	GetIcmpStatisticsEx	Not Bound
C	N/A	113 (0x0071)	GetIpStatisticsEx	Not Bound
C	N/A	137 (0x0089)	GetTcpStatisticsEx	Not Bound
C	N/A	144 (0x0090)	GetUdpStatisticsEx	Not Bound
C	N/A	181 (0x00B5)	InternalGetBoundTcp6EndpointTable	Not Bound
C	N/A	182 (0x00B6)	InternalGetBoundTcpEndpointTable	Not Bound
C	N/A	199 (0x00C7)	InternalGetTcp6Table2	Not Bound
C	N/A	200 (0x00C8)	InternalGetTcp6TableWithOwnerModule	Not Bound
C	N/A	204 (0x00CC)	InternalGetTcpTable2	Not Bound
C	N/A	206 (0x00CE)	InternalGetTcpTableWithOwnerModule	Not Bound
C	N/A	209 (0x00D1)	InternalGetUdp6Table2	Not Bound
C	N/A	214 (0x00D6)	InternalGetUdpTable2	Not Bound



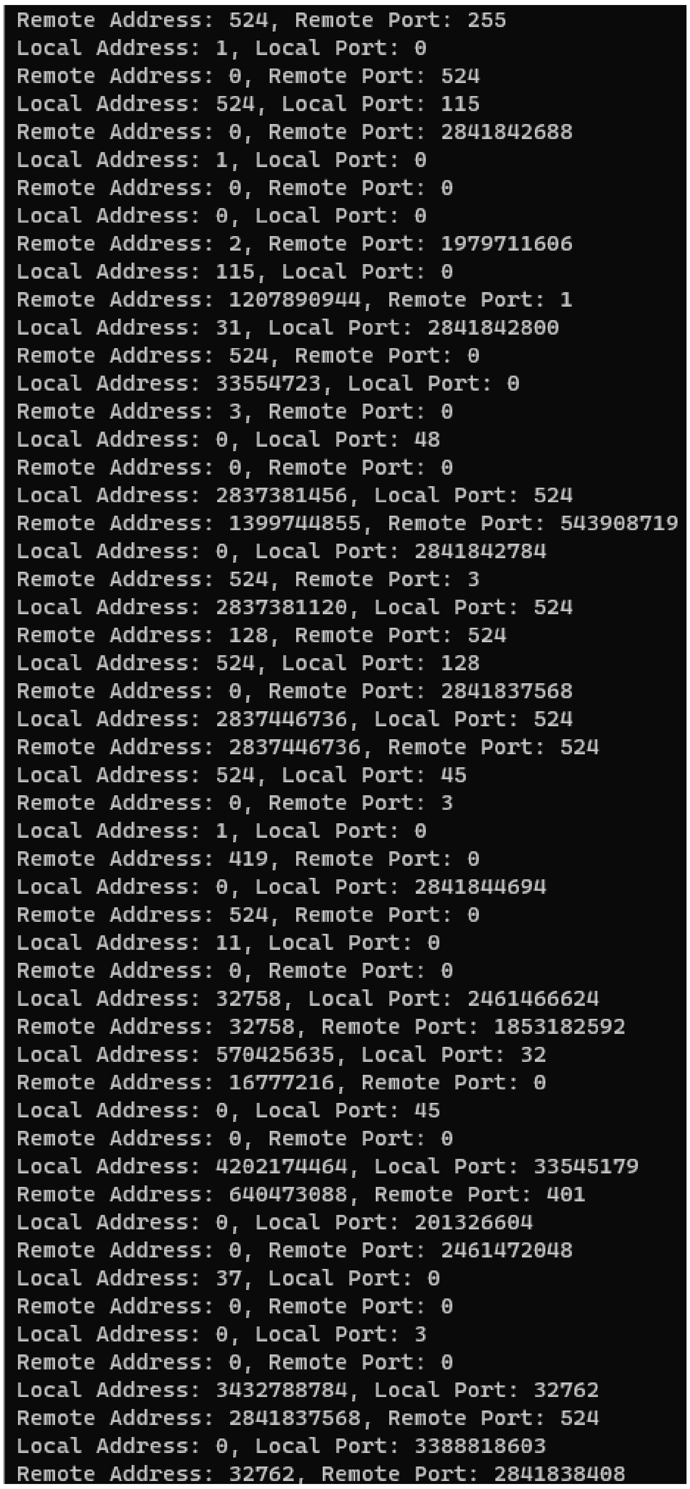
Dependency Walker - [NETSTAT]

```
NETSTAT.EXE
    MSVCRT.DLL
    NTDLL.DLL
    NSI.DLL
    IPHLPAPI.DLL
    API-MS-WIN-CORE-STRING-L1-1-0.DLL
    API-MS-WIN-CORE-HEAP-L1-1-0.DLL
    API-MS-WIN-CORE-PROCESSTHREADS-L1-1-0.DLL
    WS2_32.DLL
        API-MS-WIN-CORE-CRT-L1-1-0.DLL
        API-MS-WIN-CORE-CRT-L2-1-0.DLL
        NTDLL.DLL
```

PI	Ordinal ^	Hint	Function	Entry Point
O#	9 (0x0009)	N/A	N/A	Not Bound
O#	14 (0x000E)	N/A	N/A	Not Bound
O#	15 (0x000F)	N/A	N/A	Not Bound
O#	115 (0x0073)	N/A	N/A	Not Bound
C	N/A	8 (0x0008)	GetHostNameW	Not Bound
C	N/A	9 (0x0009)	GetNameInfoW	Not Bound

I found a couple of functions that looked promising and started to hook them via trial and error… to my surprise most of them were unused when I tried to call netstat with different flags. This was tedious and time consuming work but then I found one function that DOES get called! It was InternalGetTcpTable2, which after some research I understood that it is undocumented API 🙁… So I started to search for a workaround and searched a lot in the internet but I only found it mentioned in 2 APT analyses, which sadly didn't provide any source code, signature or anything useful. So I Tried the next improvised workaround, hope that the signature of InternalGetTcpTable2 is the same as the documented counterpart GetTcpTable2.

I coded a prototype of InternalGetTcpTable2 following the similar GetTcpTable2 and tried to hook my newly implemented function to InternalGetTcpTable2 and it worked! I could print a message to the screen through my new hooked function.
But here our luck concludes.
GetTcpTable2 fills the provided `TcpTable` with information based on a struct “`PMIB_TCPTABLE2`”. I implemented some code to print the TcpTable that worked fine on a call for GetTcpTable2, however when the print function was called on a TcpTable that was filled with the undocumented counterpart, it outputted weird numbers instead of ports and addresses:



```
Remote Address: 524, Remote Port: 255
Local Address: 1, Local Port: 0
Remote Address: 0, Remote Port: 524
Local Address: 524, Local Port: 115
Remote Address: 0, Remote Port: 2841842688
Local Address: 1, Local Port: 0
Remote Address: 0, Remote Port: 0
Local Address: 0, Local Port: 0
Remote Address: 2, Remote Port: 1979711606
Local Address: 115, Local Port: 0
Remote Address: 1207890944, Remote Port: 1
Local Address: 31, Local Port: 2841842800
Remote Address: 524, Remote Port: 0
Local Address: 33554723, Local Port: 0
Remote Address: 3, Remote Port: 0
Local Address: 0, Local Port: 48
Remote Address: 0, Remote Port: 0
Local Address: 2837381456, Local Port: 524
Remote Address: 1399744855, Remote Port: 543908719
Local Address: 0, Local Port: 2841842784
Remote Address: 524, Remote Port: 3
Local Address: 2837381120, Local Port: 524
Remote Address: 128, Remote Port: 524
Local Address: 524, Local Port: 128
Remote Address: 0, Remote Port: 2841837568
Local Address: 2837446736, Local Port: 524
Remote Address: 2837446736, Remote Port: 524
Local Address: 524, Local Port: 45
Remote Address: 0, Remote Port: 3
Local Address: 1, Local Port: 0
Remote Address: 419, Remote Port: 0
Local Address: 0, Local Port: 2841844694
Remote Address: 524, Remote Port: 0
Local Address: 11, Local Port: 0
Remote Address: 0, Remote Port: 0
Local Address: 32758, Local Port: 2461466624
Remote Address: 32758, Remote Port: 1853182592
Local Address: 570425635, Local Port: 32
Remote Address: 16777216, Remote Port: 0
Local Address: 0, Local Port: 45
Remote Address: 0, Remote Port: 0
Local Address: 4202174464, Local Port: 33545179
Remote Address: 640473088, Remote Port: 401
Local Address: 0, Local Port: 201326604
Remote Address: 0, Remote Port: 2461472048
Local Address: 37, Local Port: 0
Remote Address: 0, Remote Port: 0
Local Address: 0, Local Port: 3
Remote Address: 0, Remote Port: 0
Local Address: 3432788784, Local Port: 32762
Remote Address: 2841837568, Remote Port: 524
Local Address: 0, Local Port: 3388818603
Remote Address: 32762, Remote Port: 2841838408
```

III.
After long hours of research I couldn't figure out what was wrong, I suppose that InternalGetTcpTable expects a different struct, different from PMIB_TCPTABLE.
Further research and reverse engineering needs to be done here - couldn't find anything as such on the internet.

I also tried to hook InternalGetTcpTable2 and fill it with GetTcpTable2 but there was no luck. The netstat command didn't output the TCP connections as expected with this replacement, which means the information provided by InternalGetTcpTable2 is crucial for its workings.

Later I tried to hook more functions that could be used in the networking space and even some functions that change the representation from Multibyte to Wide Char and vice versa, all those functions seem to not get called during netstat.exe execution.

IV.
As a final product I managed to remove most of the TCP connections as a malicious activity, thus it will cover the IP of the attacker. However this change will draw the attention of the defender and he will see that the output is corrupted (most of the TCP connections are not present) and raise his suspicion.

V.
For later research I suggest reverse engineering NETSTAT.EXE to understand how it works, and what API's it uses. Also, reverse engineer the InternalGetTcpTable2 to understand what went wrong in my attack and document the undocumented function.

** source files for DLLInjector.exe, IATHooking.dll and HideInject.exe are provided.